IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of :	)	
	)	
James HAKEWILL, et al.	)	Group Art Unit: 2123
	)	
Serial No.: 09/418,663	)	Examiner: E. Garcia-Otero
	)	
Filed: October 14, 1999	)	
	)	
For: METHOD AND APPARATUS FOR MANAGING THE CONFIGURATION AND FUNCTIONALITY OF A SEMICONDUCTOR DESIGN	)	

Assistant Commissioner for Patents
Washington, D.C. 20231

Sir:

DECLARATION OF JAMES HAKEWILL UNDER 37 C.F.R. § 1.132

I, James Hakewill, a citizen of the United Kingdom, hereby declare and state as follows:

1. I am one of the co-inventors in the above-referenced patent application. I have been employed by ARC International of its predecessor companies for over twelve years. During that time, I have been involved in the design, development and testing of the ARC products referenced in the Declaration of Peter Hutton filed in the above-referenced patent application ("ARC Product").

2. The ARC Product (the versions from 1998 through the present time) identified by Peter Hutton correlates to at least the currently pending independent claims of the above-captioned application in the manner demonstrated below.

3. I am going to reference three documents produced and prepared by ARC in the 1998 timeframe related to its commercial products. The first is a Powerpoint presentation document entitled "ARC Introduction to ARC VHDL," dated February 12, 1998 related to

Version 2.1.2 of the ARC Product (the documentation was version 1.5 of the documentation related to this version of the ARC Product. That document is attached as Exhibit A to my declaration and has had page numbers A-1 through A-58 added for reference purposes only (the original did not have page numbers printed).

4. The second is a document entitled "ARC Configuration Wizard User Guide," with a copyright date of 1998 and indicating a document revision of v 1.5. That document is attached as Exhibit B.

5. The third is a document entitled "ARC System Builder User Guide," with a copyright date of 1998 and indicating a document revision of v 1.9. That document is attached as Exhibit C.

6. The following chart compares the independent claims to the ARC Product and provides a citation to the ARC documents where appropriate.

a. Independent Claim 12

Claim Recitation	ARC Product
12. An integrated circuit, fabricated using the method comprising: creating a customized description language model of an integrated circuit design by:	The ARC Product was designed to generate designs for fabricating integrated circuits. *See, e.g.,* Exhibit C generally.
receiving one or more inputs from a user for at least one customized parameter of the integrated circuit;	ARC products receive inputs from a user for customized parameters of an integrated circuit through either an interactive graphical wizard or a command line interface. *See,* Exhibit A, pages A-13-A-20 (graphical user interface) and A-21 (command line based system); Exhibit B, Section 3.4, pages 11-19; and Exhibit C, Section 3, pages 9-28 (providing a 'walkthrough' of an example run of the system builder showing the menu selections chosen to build a customized microprocessor design).

Claim Recitation	ARC Product
receiving an identification of a location of one or more library files that provide at least one prototype description and at least one extension logic description for the integrated circuit for which a model is being generated; and	In the ARC Product, a user identifies one or more library files stored in the "release" where the prototype description and extension logic description file(s) may be found. *See* Exhibit A, at page A-11 and Exhibit C at Section 2.1 – 'Setup' – page 7. Both these sections describe how to set the 'ARCHOME' environment variable to point to the location of the "arc install tree," whose structure contains the installed prototype and extension libraries.
generating through an automated process a customized description language model based on at least one customized parameter, the at least one prototype description, and the at least one extension logic description, the automated process including the acts of reading at least one prototype description and modifying the at least one prototype description by substituting values in the at least one prototype description or merging additional descriptions based on the at least one customized parameter;	The ARC Product takes the customized parameter, prototype description and extension logic description and acts to read the description and modify the prototype by substituting values or merging additional descriptions. Exhibit B indicates that upon identification of library files and other information, the user may initiate the build process to begin the automated building process performed by the ARC software. *See* Exhibit B, Section 3.1, page 8. Similarly, Exhibit C indicates that upon specifying a target path for the result and initiating the build, a result is produced as shown and explained. *See* Exhibit C, Section 3.1, page 14.
generating a netlist which is descriptive of the circuitry of said integrated circuit;	After the build process has taken place in the ARC product, a design is built in the target directory, consisting of RTL/behavioral-level VHDL modules, and structural-level VHDL modules. The RTL/behavioral-level modules describe the function of the circuit, and the structural-level modules describe how the RTL/behavioral modules are interconnected. *See* Exhibit A at page A-24.
compiling said netlist and said hardware description model to produce a compiled integrated circuit design;	The ARC product user would then use the netlist and hardware description model to produce a compiled integrated circuit design. *See* Exhibit A, at pages A-38-A-50.

Claim Recitation	ARC Product
fabricating at least one mask or FPGA configuration file representing said compiled integrated circuit design;	The ARC product user (or another entity if desired) would then fabricate a mask or FPGA configuration file. *See* Exhibit A at page 56, for an example of FPGA configuration file generation.
fabricating said integrated circuit using said at least one mask or FPGA configuration file;	The ARC product user (or another entity if desired) would then fabricate an integrated circuit using the mask or FPGA configuration file
wherein said act of creating is performed at a high level of abstraction.	The ARC product performs these acts at a high level of abstraction.

b. Independent Claim 18

Claim Recitation	ARC Product
18. An apparatus adapted to generate integrated circuit designs, comprising:	The ARC Product is an apparatus adapted to generate integrated circuit designs.
a processor capable of running a computer program;	The ARC Product operates on a processor capable of running a computer program.
a storage device operatively coupled to said processor, said storage device being capable of storing at least a portion of a computer program;	The ARC Product operates on a computer that is associated with a storage device as recited.
an input device, operatively coupled to said processor capable of receiving input from a user and transmitting said input to said processor; and	The ARC Product operates on a computer that is associated with an input device as recited.
a computer program resident at least in part on said storage device, said computer program adapted perform the following acts:	The ARC Product includes a computer program stored on the storage device that performs the recited acts as identified below.

Claim Recitation	ARC Product
receiving one or more inputs from a user for at least one customized parameter of the integrated circuit;	The ARC Product receives inputs from a user for customized parameters of an integrated circuit through either an interactive graphical wizard or a command line interface. *See,* Exhibit A, pages A-13-A-20 (graphical user interface) and A-21 (command line based system); Exhibit B, Section 3.4, pages 11-19; and Exhibit C, Section 3, pages 9-28 (providing a 'walkthrough' of an example run of the system builder showing the menu selections chosen to build a customized microprocessor design).
receiving an identification of a location of one or more library files that provide at least one prototype description and at least one extension logic description for the integrated circuit for which a model is being generated; and	In the ARC product, a user identifies one or more library files stored in the "release" where the prototype description and extension logic description file(s) may be found. *See* Exhibit A, at page A-11 and Exhibit C at Section 2.1 – 'Setup' – page 7. Both these sections describe how to set the 'ARCHOME' environment variable to point to the location of the "arc install tree," whose structure contains the installed prototype and extension libraries.
generating a customized description language model based on at least one customized parameter, the at least one prototype description, and the at least one extension logic description including the acts of reading at least one prototype description and modifying the at least one prototype description by substituting values in the at least one prototype description or merging additional descriptions based on the at least one customized parameter.	The ARC product takes the customized parameter, prototype description and extension logic description and acts to read the description and modify the prototype by substituting values or merging additional descriptions. Exhibit B indicates that upon identification of library files and other information, the user may initiate the build process to begin the automated building process performed by the ARC software. *See* Exhibit B, Section 3.1, page 8. Similarly, Exhibit C indicates that upon specifying a target path for the result and initiating the build, a result is produced as shown and explained. *See* Exhibit C, Section 3.1, page 14.

c. Independent Claim 40

Claim Recitation	ARC Product
40. A system for generating integrated circuit designs at a high level of abstraction comprising:	The ARC Product may operate as part of a system for generating integrated circuit designs at a high level of abstraction.
a processor;	The ARC Product operates on a processor.
a storage device in data communication with said processor, said storage device being capable of storing and retrieving a computer program; and	The ARC Product operates on a computer that is associated with a storage device as recited.
a computer program stored within said storage device and adapted to run on said processor, said computer program comprising;	The ARC Product includes a computer program stored on the storage device that performs the recited acts as identified below.
an input receiving module that receives one or more inputs from a user for at least one customized parameter of an integrated circuit device, the at least one customized parameter comprising a parameter selected from the group comprising a custom instruction, a cache configuration, a memory interface configuration and a system architecture configuration;	The ARC Product operates on a computer that is associated with an input receiving module. ARC products received inputs from a user for customized parameters of an integrated circuit device through either an interactive graphical wizard or a command line interface. *See,* Exhibit A, pages A-13-A-20 (graphical user interface) and A-21 (command line based system); Exhibit B, Section 3.4, pages 11-19; and Exhibit C, Section 3, pages 9-28 (providing a 'walkthrough' of an example run of the system builder showing the menu selections chosen to build a customized microprocessor design).

Claim Recitation	ARC Product
a library file receiving module that receives an identification of a location of one or more library files that provide at least one prototype description and at least one extension logic description for the integrated circuit device for which a model is being generated; and	In the ARC product, a library file receiving module receives an identification of one or more library files stored in the "release" where the prototype description and extension logic description file(s) may be found. *See* Exhibit A, at page A-11 and Exhibit C at Section 2.1 – 'Setup' – page 7. Both these sections describe how to set the 'ARCHOME' environment variable to point to the location of the "arc install tree," whose structure contains the installed prototype and extension libraries.
a generation module that generates a customized description language model based on at least one customized parameter, the at least one prototype description, and the at least one extension logic description through acts including reading at least one prototype description and modifying the at least one prototype description by substituting values in the at least one prototype description or merging additional descriptions based on the at least one customized parameter.	The ARC product takes the customized parameter, prototype description and extension logic description and acts to read the description and modify the prototype by substituting values or merging additional descriptions. Exhibit B indicates that upon identification of library files and other information, the user may initiate the build process to begin the automated building process performed by the ARC software. *See* Exhibit B, Section 3.1, page 8. Similarly, Exhibit C indicates that upon specifying a target path for the result and initiating the build, a result is produced as shown and explained. *See* Exhibit C, Section 3.1, page 14.

d. Independent Claim 47

Claim Recitation	ARC Product
47. A method of generating the design of an integrated circuit rendered in a hardware description language, said method being performed at a high level of abstraction and comprising the acts of:	The ARC Product enables its users to perform a method of generating a design as stated.
selecting a process technology;	The ARC Product user selects a desired process technology.

Claim Recitation	ARC Product
receiving one or more inputs from a user for at least one customized parameter of the integrated circuit, including at least one parameter selected from the group comprising: (i) processor instructions; (ii) cache configuration; (iii) memory interface configuration; and (iv) system architecture configuration;	ARC products receive inputs from a user for customized parameters through either an interactive graphical wizard or a command line interface. *See,* Exhibit A, pages A-13-A-20 (graphical user interface) and A-21 (command line based system); Exhibit B, Section 3.4, pages 11-19; and Exhibit C, Section 3, pages 9-28 (providing a 'walkthrough' of an example run of the system builder showing the menu selections chosen to build a customized microprocessor design). The parameters that the user may select include processor instructions, cache configuration, memory interface configuration or system architecture configuration. *Id.*
receiving an identification of a location of at least one library file that provides at least one prototype description and at least one extension logic description for the integrated circuit for which a model is being generated; and	In the ARC product, a user identifies one or more library files stored in the "release" where the prototype description and extension logic description file(s) may be found. *See* Exhibit A, at page A-11 and Exhibit C at Section 2.1 – 'Setup' – page 7. Both these sections describe how to set the 'ARCHOME' environment variable to point to the location of the "arc install tree," whose structure contains the installed prototype and extension libraries.

Claim Recitation	ARC Product
generating through an automated process a customized description language model based on at least one customized parameter, the at least one prototype description, and the at least one extension logic description, the automated process including the acts of reading at least one prototype description and modifying the at least one prototype description by substituting values in the at least one prototype description or merging additional descriptions based on the at least one customized parameter.	The ARC product takes the customized parameter, prototype description and extension logic description and acts to read the description and modify the prototype by substituting values or merging additional descriptions. Exhibit B indicates that upon identification of library files and other information, the user may initiate the build process to begin the automated building process performed by the ARC software. *See* Exhibit B, Section 3.1, page 8. Similarly, Exhibit C indicates that upon specifying a target path for the result and initiating the build, a result is produced as shown and explained. *See* Exhibit C, Section 3.1, page 14.

e. Independent Claim 48

Claim Recitation	ARC Product
48. A system for generating integrated circuit designs at a high level of abstraction, comprising:	The ARC Product may operate as part of a system for generating integrated circuit designs at a high level of abstraction.
means for processing digital data;	The ARC Product operates on a processor.
means for data storage in data communication with said processor means, said means for data storage being capable of storing and retrieving a computer program; and	The ARC Product operates on a computer that is associated with a storage device as recited.
a computer program stored within said means for data storage and adapted to run on said processor means, said computer program comprising:	The ARC Product includes a computer program stored on the storage device that performs the recited acts as identified below.
means for selecting a process technology;	The ARC Product user may select a process technology and indicate that selection to the ARC Product

Claim Recitation	ARC Product
an input receiving module that receives one or more inputs from a user for at least one customized parameter of the integrated circuit device, including at least one parameter selected from the group comprising a cache configuration, a memory interface configuration, and a system architecture configuration;	The ARC Product operates on a computer that is associated with an input receiving module. ARC products received inputs from a user for customized parameters through either an interactive graphical wizard or a command line interface. *See,* Exhibit A, pages A-13-A-20 (graphical user interface) and A-21 (command line based system); Exhibit B, Section 3.4, pages 11-19; and Exhibit C, Section 3, pages 9-28 (providing a 'walkthrough' of an example run of the system builder showing the menu selections chosen to build a customized microprocessor design).
a library file receiving module that receives an identification of a location of one or more library files that provide at least one prototype description and at least one extension logic description for the integrated circuit device for which a model is being generated; and	In the ARC product, a library file receiving module receives an identification of one or more library files stored in the "release" where the prototype description and extension logic description file(s) may be found. *See* Exhibit A, at page A-11 and Exhibit C at Section 2.1 – 'Setup' – page 7. Both these sections describe how to set the 'ARCHOME' environment variable to point to the location of the "arc install tree," whose structure contains the installed prototype and extension libraries.

Claim Recitation	ARC Product
a generation module that generates a customized description language model based on at least one customized parameter, the at least one prototype description, and the at least one extension logic description through acts including reading one or more prototype description and modifying the one or more prototype description by substituting values in the one or more prototype description or merging additional descriptions based on the at least one customized parameter.	The ARC product takes the customized parameter, prototype description and extension logic description and acts to read the description and modify the prototype by substituting values or merging additional descriptions. Exhibit B indicates that upon identification of library files and other information, the user may initiate the build process to begin the automated building process performed by the ARC software. *See* Exhibit B, Section 3.1, page 8. Similarly, Exhibit C indicates that upon specifying a target path for the result and initiating the build, a result is produced as shown and explained. *See* Exhibit C, Section 3.1, page 14.

f. Independent Claim 75

Claim Recitation	ARC Product
75. A computer-implemented method of generating the design of an integrated circuit at a high level of abstraction using a description language, comprising the acts of:	The ARC Product enables a computer-implemented method of generating the stated design.
providing an existing processor core configuration;	The ARC Product provides an existing processor core configuration.

Claim Recitation	ARC Product
receiving one or more inputs from a user for at least one customized parameter of the existing processor core configuration for the integrated circuit device, the input being selected from a editing a first file specific to the design, said editing comprising selecting a constrained set of input parameters associated with said configuration, said parameters comprising: (i) at least one custom instruction; (ii) a cache configuration; and (iii) a memory interface configuration;	ARC products receive inputs from a user for customized parameters through either an interactive graphical wizard or a command line interface. *See,* Exhibit A, pages A-13-A-20 (graphical user interface) and A-21 (command line based system); Exhibit B, Section 3.4, pages 11-19; and Exhibit C, Section 3, pages 9-28 (providing a 'walkthrough' of an example run of the system builder showing the menu selections chosen to build a customized microprocessor design). The parameters a user may provide include at least one custom instruction, a cache configuration and a memory interface configuration.
receiving an identification of a location of one or more library files that provide at least one prototype description and at least one extension logic description for the integrated circuit device for which a model is being generated; and	In the ARC product, a user identifies one or more library files stored in the "release" where the prototype description and extension logic description file(s) may be found. *See* Exhibit A, at page A-11 and Exhibit C at Section 2.1 – 'Setup' – page 7. Both these sections describe how to set the 'ARCHOME' environment variable to point to the location of the "arc install tree," whose structure contains the installed prototype and extension libraries.

Claim Recitation	ARC Product
generating through an automated process a customized description language model based on at least one customized parameter, the at least one prototype description, and the at least one extension logic description, the automated process including the acts of reading at least one prototype description and modifying the at least one prototype description by substituting values in the at least one prototype description or merging additional descriptions based on the at least one customized parameter.	The ARC product takes the customized parameter, prototype description and extension logic description and acts to read the description and modify the prototype by substituting values or merging additional descriptions. Exhibit B indicates that upon identification of library files and other information, the user may initiate the build process to begin the automated building process performed by the ARC software. *See* Exhibit B, Section 3.1, page 8. Similarly, Exhibit C indicates that upon specifying a target path for the result and initiating the build, a result is produced as shown and explained. *See* Exhibit C, Section 3.1, page 14.

g. Independent Claim 76

Claim Recitation	ARC Product
76. A method of generating an integrated circuit design at a high level of abstraction, comprising:	The ARC Produce enables such a method.
providing a user with a plurality of optional inputs, including the ability to generate a customized hardware description language code instruction;	ARC products receive inputs from a user for customized parameters through either an interactive graphical wizard or a command line interface. *See,* Exhibit A, pages A-13-A-20 (graphical user interface) and A-21 (command line based system); Exhibit B, Section 3.4, pages 11-19; and Exhibit C, Section 3, pages 9-28 (providing a 'walkthrough' of an example run of the system builder showing the menu selections chosen to build a customized microprocessor design).
selecting at least one of said plurality of optional inputs;	The user of ARC may select inputs.
selecting at least one cache configuration;	The user may, if desired, select at least one cache configuration.
defining at least one memory interface;	The user may, if desired, define at least one memory interface.

Claim Recitation	ARC Product
generating through an automated process a customized description language model based on at least one optional input, cache configuration, and memory interface customized parameter, the automated process including the acts of reading at least one prototype description, modifying the at least one prototype description by substituting values in the at least one prototype description or merging additional descriptions based on the at least one optional input, cache configuration and memory interface, and incorporating any customized hardware description language code instructions.	The ARC product takes the customized parameter, prototype description and extension logic description and acts to read the description and modify the prototype by substituting values or merging additional descriptions. Exhibit B indicates that upon identification of library files and other information, the user may initiate the build process to begin the automated building process performed by the ARC software. *See* Exhibit B, Section 3.1, page 8. Similarly, Exhibit C indicates that upon specifying a target path for the result and initiating the build, a result is produced as shown and explained. *See* Exhibit C, Section 3.1, page 14.

h. Independent Claim 77

Claim Recitation	ARC Product
77. A description language model of an integrated circuit design generated at a high level of abstraction using the method comprising:	The ARC Product enables the user to generate such a model.
receiving one or more inputs from a user for at least one customized parameter of the existing processor core configuration for the integrated circuit device, the input being selected from a plurality of input parameters associated with said design, said parameters comprising:(i) at least one extension instruction; and (ii) a cache configuration;	ARC products receive inputs from a user for customized parameters through either an interactive graphical wizard or a command line interface. *See,* Exhibit A, pages A-13-A-20 (graphical user interface) and A-21 (command line based system); Exhibit B, Section 3.4, pages 11-19; and Exhibit C, Section 3, pages 9-28 (providing a 'walkthrough' of an example run of the system builder showing the menu selections chosen to build a customized microprocessor design). The parameters a user may select include at least one extension instruction and a cache configuration.

Claim Recitation	ARC Product
defining the location of at least one library file that provides at least one prototype description and at least one extension logic description for the integrated circuit device for which a model is being generated; and	In the ARC product, a user identifies one or more library files stored in the "release" where the prototype description and extension logic description file(s) may be found. *See* Exhibit A, at page A-11 and Exhibit C at Section 2.1 – 'Setup' – page 7. Both these sections describe how to set the 'ARCHOME' environment variable to point to the location of the "arc install tree," whose structure contains the installed prototype and extension libraries.
generating through an automated process a customized description language model based on at least one customized parameter, the at least one prototype description, and the at least one extension logic description, the automated process including the acts of reading at least one prototype description and modifying the at least one prototype description by substituting values in the at least one prototype description or merging additional descriptions based on the at least one customized parameter.	The ARC product takes the customized parameter, prototype description and extension logic description and acts to read the description and modify the prototype by substituting values or merging additional descriptions. Exhibit B indicates that upon identification of library files and other information, the user may initiate the build process to begin the automated building process performed by the ARC software. *See* Exhibit B, Section 3.1, page 8. Similarly, Exhibit C indicates that upon specifying a target path for the result and initiating the build, a result is produced as shown and explained. *See* Exhibit C, Section 3.1, page 14.

i. Independent Claim 78

Claim Recitation	ARC Product
78. A method of generating an extended processor design at a high level of abstraction, comprising:	The ARC Product user may perform such a method.
providing the user with a basecase processor core configuration having a base instruction set;	The ARC Product enables the provision of a basecase processor core configuration.
providing a user with a plurality of optional instructions adaptable for use with said basecase core;	The ARC Product provides a user with optional instructions.

Claim Recitation	ARC Product
receiving the selection of at least one of said plurality of optional instructions;	ARC products receive inputs from a user for customized parameters through either an interactive graphical wizard or a command line interface. *See,* Exhibit A, pages A-13-A-20 (graphical user interface) and A-21 (command line based system); Exhibit B, Section 3.4, pages 11-19; and Exhibit C, Section 3, pages 9-28 (providing a 'walkthrough' of an example run of the system builder showing the menu selections chosen to build a customized microprocessor design).
receiving the selection of at least one cache configuration;	The ARC Product user may, if desired, select at least one cache configuration.
receiving an identification of a location of one or more library files that provide at least one basecase description and at least one extension logic description for the integrated circuit device for which a model is being generated;	In the ARC product, a user identifies one or more library files stored in the "release" where the prototype description and extension logic description file(s) may be found. *See* Exhibit A, at page A-11 and Exhibit C at Section 2.1 – 'Setup' – page 7. Both these sections describe how to set the 'ARCHOME' environment variable to point to the location of the "arc install tree," whose structure contains the installed prototype and extension libraries.

Claim Recitation	ARC Product
generating through an automated process a customized description language model based on at least one optional instruction, the at least one basecase description, and the at least one extension logic description, the automated process including the acts of reading at least one basecase description and modifying the at least one basecase description by substituting values in the at least one basecase description or merging additional descriptions based on the at least one customized parameter; and	The ARC product takes the customized parameter, prototype description and extension logic description and acts to read the description and modify the prototype by substituting values or merging additional descriptions. Exhibit B indicates that upon identification of library files and other information, the user may initiate the build process to begin the automated building process performed by the ARC software. *See* Exhibit B, Section 3.1, page 8. Similarly, Exhibit C indicates that upon specifying a target path for the result and initiating the build, a result is produced as shown and explained. *See* Exhibit C, Section 3.1, page 14.
wherein said plurality of optional instructions and cache configurations are constrained so as to ensure the functionality of said processor design irrespective of the user's selections.	The ARC Product may operate in this manner.

j. Independent Claim 79

Claim Recitation	ARC Product
79. A computer-implemented method of generating a customized description language model of a integrated circuit design including at least one of a microprocessor or microprocessor peripheral device comprising the following acts performed by a computer process:	The ARC Product enables such a method.

Claim Recitation	ARC Product
receiving one or more inputs from a user for at least one customized parameter of the microprocessor or microprocessor peripheral;	ARC products receive inputs from a user for customized parameters through either an interactive graphical wizard or a command line interface. *See,* Exhibit A, pages A-13-A-20 (graphical user interface) and A-21 (command line based system); Exhibit B, Section 3.4, pages 11-19; and Exhibit C, Section 3, pages 9-28 (providing a 'walkthrough' of an example run of the system builder showing the menu selections chosen to build a customized microprocessor design).
receiving an identification of a location of one or more library files that provide at least one prototype description and at least one extension logic description for the microprocessor or microprocessor peripheral for which a model is being generated; and	ARC products receive inputs from a user for customized parameters through either an interactive graphical wizard or a command line interface. *See,* Exhibit A, pages A-13-A-20 (graphical user interface) and A-21 (command line based system); Exhibit B, Section 3.4, pages 11-19; and Exhibit C, Section 3, pages 9-28 (providing a 'walkthrough' of an example run of the system builder showing the menu selections chosen to build a customized microprocessor design).
generating through an automated process a customized description language model based on the least one customized parameter, the at least one prototype description, and the at least one extension logic description, the automated process including the acts of reading at least one of the prototype descriptions and modifying the at least one prototype description by substituting values in the at least one prototype description or merging additional descriptions based on the at least one customized parameter.	ARC products receive inputs from a user for customized parameters through either an interactive graphical wizard or a command line interface. *See,* Exhibit A, pages A-13-A-20 (graphical user interface) and A-21 (command line based system); Exhibit B, Section 3.4, pages 11-19; and Exhibit C, Section 3, pages 9-28 (providing a 'walkthrough' of an example run of the system builder showing the menu selections chosen to build a customized microprocessor design).

k. Independent Claim 85

Claim Recitation	ARC Product
85. A method of designing an integrated circuit design including at least one of a microprocessor or microprocessor peripheral device comprising the acts of:	The ARC Product enables such a method of design.
receiving one or more inputs from a user for at least one customized parameter of the microprocessor or microprocessor peripheral;	ARC products receive inputs from a user for customized parameters through either an interactive graphical wizard or a command line interface. *See,* Exhibit A, pages A-13-A-20 (graphical user interface) and A-21 (command line based system); Exhibit B, Section 3.4, pages 11-19; and Exhibit C, Section 3, pages 9-28 (providing a 'walkthrough' of an example run of the system builder showing the menu selections chosen to build a customized microprocessor design).
receiving an identification of a location of one or more library files that provide at least one prototype description and at least one extension logic description for the microprocessor or microprocessor peripheral for which a model is being generated;	ARC products receive inputs from a user for customized parameters through either an interactive graphical wizard or a command line interface. *See,* Exhibit A, pages A-13-A-20 (graphical user interface) and A-21 (command line based system); Exhibit B, Section 3.4, pages 11-19; and Exhibit C, Section 3, pages 9-28 (providing a 'walkthrough' of an example run of the system builder showing the menu selections chosen to build a customized microprocessor design).

Claim Recitation	ARC Product
generating through an automated process a customized description language model based on the least one customized parameter, the at least one prototype description, and the at least one extension logic description, the automated process including the acts of reading at least one prototype description and modifying the at least one prototype description by substituting values in the at least one prototype description or merging additional descriptions based on the at least one customized parameter;	ARC products receive inputs from a user for customized parameters through either an interactive graphical wizard or a command line interface. *See,* Exhibit A, pages A-13-A-20 (graphical user interface) and A-21 (command line based system); Exhibit B, Section 3.4, pages 11-19; and Exhibit C, Section 3, pages 9-28 (providing a 'walkthrough' of an example run of the system builder showing the menu selections chosen to build a customized microprocessor design).
testing the customized description language through simulation or synthesis; and	ARC products users may then test the customized description language.
if a different testing outcome is desired, receiving an identification of one or more input for at least one customized parameter and generating a second customized description language model.	ARC Product users may then reuse the system to identify different inputs to generate a second model.

l. Independent Claim 91

Claim Recitation	ARC Product
91. An apparatus that generates a customized description language model of an integrated circuit design including at least one of a microprocessor or microprocessor peripheral device comprising:	The ARC Product provides such an apparatus.
an input module that receives one or more inputs from a user for at least one customized parameter of the microprocessor or microprocessor peripheral;	ARC products receive inputs from a user for customized parameters through either an interactive graphical wizard or a command line interface. *See,* Exhibit A, pages A-13-A-20 (graphical user interface) and A-21 (command line based system); Exhibit B, Section 3.4, pages 11-19; and Exhibit C, Section 3, pages 9-28 (providing a 'walkthrough' of an example run of the system builder showing the menu selections chosen to build a customized microprocessor design).

Claim Recitation	ARC Product
a library file module that receives an identification of a location of one or more library files that provide at least one prototype description and at least one extension logic description for the integrated circuit device for which a model is being generated;	In the ARC product, a library file receiving module receives an identification of one or more library files stored in the "release" where the prototype description and extension logic description file(s) may be found. *See* Exhibit A, at page A-11 and Exhibit C at Section 2.1 – 'Setup' – page 7. Both these sections describe how to set the 'ARCHOME' environment variable to point to the location of the "arc install tree," whose structure contains the installed prototype and extension libraries.
a description language model generator that generates a customized description language model based on the least one customized parameter, the at least one prototype description and the at least one extension logic description through an automated process that reads at least one prototype description and modifies the at least one prototype description by substituting values in the at least one prototype description or merging additional descriptions based on the at least one customized parameter; and	The ARC product takes the customized parameter, prototype description and extension logic description and acts to read the description and modify the prototype by substituting values or merging additional descriptions. Exhibit B indicates that upon identification of library files and other information, the user may initiate the build process to begin the automated building process performed by the ARC software. *See* Exhibit B, Section 3.1, page 8. Similarly, Exhibit C indicates that upon specifying a target path for the result and initiating the build, a result is produced as shown and explained. *See* Exhibit C, Section 3.1, page 14.
wherein the customized description language model includes both functional and structural description language descriptions for the microprocessor or microprocessor peripheral.	The ARC Product generates both functional and structural description language descriptions.

m. Independent Claim 97

Claim Recitation	ARC Product
97. An apparatus that generates a customized description language model of an integrated circuit design including at least one of a microprocessor or microprocessor peripheral device comprising:	The ARC Product provides such an appartus.

Claim Recitation	ARC Product
an input module that receives one or more inputs from a user for at least one customized parameter of the microprocessor or microprocessor peripheral;	ARC products receive inputs from a user for customized parameters through either an interactive graphical wizard or a command line interface. *See,* Exhibit A, pages A-13-A-20 (graphical user interface) and A-21 (command line based system); Exhibit B, Section 3.4, pages 11-19; and Exhibit C, Section 3, pages 9-28 (providing a 'walkthrough' of an example run of the system builder showing the menu selections chosen to build a customized microprocessor design).
a library file module that receives an identification of a location of one or more library files that provide at least one prototype description and at least one extension logic description for the integrated circuit device for which a model is being generated;	In the ARC product, a library file receiving module receives an identification of one or more library files stored in the "release" where the prototype description and extension logic description file(s) may be found. *See* Exhibit A, at page A-11 and Exhibit C at Section 2.1 – 'Setup' – page 7. Both these sections describe how to set the 'ARCHOME' environment variable to point to the location of the "arc install tree," whose structure contains the installed prototype and extension libraries.
a description language model generator that generates a customized description language model based on the least one customized parameter, the at least one prototype description and the at least one extension logic description through an automated process that reads at least one prototype description and modifies the at least one prototype description by substituting values in the at least one prototype description or merging additional descriptions based on the at least one customized parameter; and	The ARC product takes the customized parameter, prototype description and extension logic description and acts to read the description and modify the prototype by substituting values or merging additional descriptions. Exhibit B indicates that upon identification of library files and other information, the user may initiate the build process to begin the automated building process performed by the ARC software. *See* Exhibit B, Section 3.1, page 8. Similarly, Exhibit C indicates that upon specifying a target path for the result and initiating the build, a result is produced as shown and explained. *See* Exhibit C, Section 3.1, page 14.

Claim Recitation	ARC Product
a feedback module that enables a user to test the customized description language through simulation or synthesis;	ARC Product users may then test the customized description language.
wherein if a different testing outcome is desired, the input receiving module enables the user to input an identification of one or more input for at least one customized parameter and the description language generator generates a second customized description language model; and	ARC Product users may then reuse the system to identify different inputs to generate a second model.
wherein the customized description language model includes both functional and structural description language descriptions for the microprocessor or microprocessor peripheral.	The ARC Product generates both functional and structural description language descriptions.

n. Independent Claim 102

Claim Recitation	ARC Product
102. A computer-implemented method of generating a customized description language model and associated test code of a integrated circuit design including at least one of a microprocessor or microprocessor peripheral device comprising the following acts performed by a computer process:	The ARC Product enables such a method.
receiving one or more inputs from a user for at least one customized parameter of the microprocessor or microprocessor peripheral;	The ARC Product receives inputs from a user for customized parameters through either an interactive graphical wizard or a command line interface. *See,* Exhibit A, pages A-13-A-20 (graphical user interface) and A-21 (command line based system); Exhibit B, Section 3.4, pages 11-19; and Exhibit C, Section 3, pages 9-28 (providing a 'walkthrough' of an example run of the system builder showing the menu selections chosen to build a customized microprocessor design).

Claim Recitation	ARC Product
generating through an automated process a customized description language model based on the least one customized parameter, the automated process including the act of modifying at least one prototype description by substituting values in the at least one prototype description or merging additional descriptions based on the at least one customized parameter;	The ARC product takes the customized parameter, prototype description and extension logic description and acts to read the description and modify the prototype by substituting values or merging additional descriptions. Exhibit B indicates that upon identification of library files and other information, the user may initiate the build process to begin the automated building process performed by the ARC software. *See* Exhibit B, Section 3.1, page 8. Similarly, Exhibit C indicates that upon specifying a target path for the result and initiating the build, a result is produced as shown and explained. *See* Exhibit C, Section 3.1, page 14.
generating through an automated process test code associated with the customized description language model based on the at least one customized parameter; and	ARC Product generates test code for users to test the customized description language.
wherein the customized description language model includes both functional and structural description language descriptions for the microprocessor or microprocessor peripheral.	The ARC Product generates both functional and structural description language descriptions.

o. Independent Claim 104

Claim Recitation	ARC Product
104. A method of designing an integrated circuit design including at least one of a microprocessor or microprocessor peripheral device comprising the acts of:	The ARC Product enables such a method.

Claim Recitation	ARC Product
receiving one or more inputs from a user for at least one customized parameter of the microprocessor or microprocessor peripheral;	The ARC product receives inputs from a user for customized parameters through either an interactive graphical wizard or a command line interface. *See,* Exhibit A, pages A-13-A-20 (graphical user interface) and A-21 (command line based system); Exhibit B, Section 3.4, pages 11-19; and Exhibit C, Section 3, pages 9-28 (providing a 'walkthrough' of an example run of the system builder showing the menu selections chosen to build a customized microprocessor design).
generating through an automated process a customized description language model based on the least one customized parameter, the automated process including the act of modifying at least one prototype description by substituting values in the at least one prototype description or merging additional descriptions based on the at least one customized parameter;	The ARC product takes the customized parameter, prototype description and extension logic description and acts to read the description and modify the prototype by substituting values or merging additional descriptions. Exhibit B indicates that upon identification of library files and other information, the user may initiate the build process to begin the automated building process performed by the ARC software. *See* Exhibit B, Section 3.1, page 8. Similarly, Exhibit C indicates that upon specifying a target path for the result and initiating the build, a result is produced as shown and explained. *See* Exhibit C, Section 3.1, page 14.
generating through an automated process test code associated with the customized description language model based on the at least one customized parameter	The ARC Product generates test code through the automated process.
testing the customized description language through simulation or synthesis using the test code generated;	ARC products users may then test the customized description language.
if a different testing outcome is desired, receiving an identification of one or more input for at least one customized parameter and generating a second customized description language model; and	ARC Product users may then reuse the system to identify different inputs to generate a second model.

Claim Recitation	ARC Product
wherein the customized description language model includes both functional and structural description language descriptions for the microprocessor or microprocessor peripheral.	The ARC Product generates both functional and structural description language descriptions.

p. Independent Claim 106

Claim Recitation	ARC Product
106. An apparatus that generates a customized description language model of an integrated circuit design including at least one of a microprocessor or microprocessor peripheral device comprising:	The ARC Product enables such an apparatus when loaded on a computer.
an input module that receives one or more inputs from a user for at least one customized parameter of the microprocessor or microprocessor peripheral;	The ARC product receives inputs from a user for customized parameters through either an interactive graphical wizard or a command line interface. *See,* Exhibit A, pages A-13-A-20 (graphical user interface) and A-21 (command line based system); Exhibit B, Section 3.4, pages 11-19; and Exhibit C, Section 3, pages 9-28 (providing a 'walkthrough' of an example run of the system builder showing the menu selections chosen to build a customized microprocessor design).
a description language model generator that generates through an automated process a customized description language model based on the least one customized parameter, the automated process including the act of modifying at least one prototype description by substituting values in the at least one prototype description or merging additional descriptions based on the at least one customized parameter;	The ARC product takes the customized parameter, prototype description and extension logic description and acts to read the description and modify the prototype by substituting values or merging additional descriptions. Exhibit B indicates that upon identification of library files and other information, the user may initiate the build process to begin the automated building process performed by the ARC software. *See* Exhibit B, Section 3.1, page 8. Similarly, Exhibit C indicates that upon specifying a target path for the result and initiating the build, a result is produced as shown and explained. *See* Exhibit C, Section 3.1, page 14.

Claim Recitation	ARC Product
a test code generator that generates through an automated process test code associated with the customized description language model based on the at least one customized parameter; and	The ARC Product provides a test code generator as recited.
wherein the customized description language model includes both functional and structural description language descriptions for the microprocessor or microprocessor peripheral.	The ARC Product generates both functional and structural description language descriptions.

q. Independent Claim 108

Claim Recitation	ARC Product
108. An apparatus that generates a customized description language model of an integrated circuit design including at least one of a microprocessor or microprocessor peripheral device comprising:	The ARC Product enables such a product when loaded on a computer.
an input module that receives one or more inputs from a user for at least one customized parameter of the microprocessor or microprocessor peripheral;	The ARC product receives inputs from a user for customized parameters through either an interactive graphical wizard or a command line interface. *See,* Exhibit A, pages A-13-A-20 (graphical user interface) and A-21 (command line based system); Exhibit B, Section 3.4, pages 11-19; and Exhibit C, Section 3, pages 9-28 (providing a 'walkthrough' of an example run of the system builder showing the menu selections chosen to build a customized microprocessor design).

Claim Recitation	ARC Product
a description language model generator that generates through an automated process a customized description language model based on the least one customized parameter, the automated process including the act of modifying at least one prototype description by substituting values in the at least one prototype description or merging additional descriptions based on the at least one customized parameter;	The ARC product takes the customized parameter, prototype description and extension logic description and acts to read the description and modify the prototype by substituting values or merging additional descriptions. Exhibit B indicates that upon identification of library files and other information, the user may initiate the build process to begin the automated building process performed by the ARC software. *See* Exhibit B, Section 3.1, page 8. Similarly, Exhibit C indicates that upon specifying a target path for the result and initiating the build, a result is produced as shown and explained. *See* Exhibit C, Section 3.1, page 14.
a test code generator that generates through an automated process test code associated with the customized description language model based on the at least one customized parameter;	The ARC Product generates test code as recited.
a feedback module that enables a user to test the customized description language through simulation or synthesis using the test code generated; and	The ARC Product may be used with a feedback module to test the model.
wherein if a different testing outcome is desired, the input receiving module enables the user to input an identification of one or more input for at least one customized parameter and the description language generator generates a second customized description language model.	The ARC Product enables a user to select an input and generate a second model.

7. All statements made herein of my own knowledge are true, and all statements made on information and belief are believed to be true. These statements were made with the knowledge that willful false statements an the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code, and that such

willful false statements may jeopardize the validity of the application or any patent issuing thereon.

James Hakewill

DATE: 7/26/2004

Introduction to ARC VHDL

Document revision - 12-Feb-98, v.1.5

Outline

- What's in the release?
- How the libraries work
- How the structural hierarchy is generated
- Installing
- Generating a customised basecase ARC
- Demonstrations

What's in the release?

- Basecase VHDL files
- Extensions VHDL files
- Tool for generating VHDL hierarchy
- ARC builder tool
- Test code and testbenches
- Documentation

VHDL notes

- IEEE 1076-1987
- Designed for synthesis using Synopsys
- Well commented
- Base type is std_ulogic
- No Synopsys DesignWare parts instantiated directly by basecase.
  - some are used by extensions

VHDL libraries

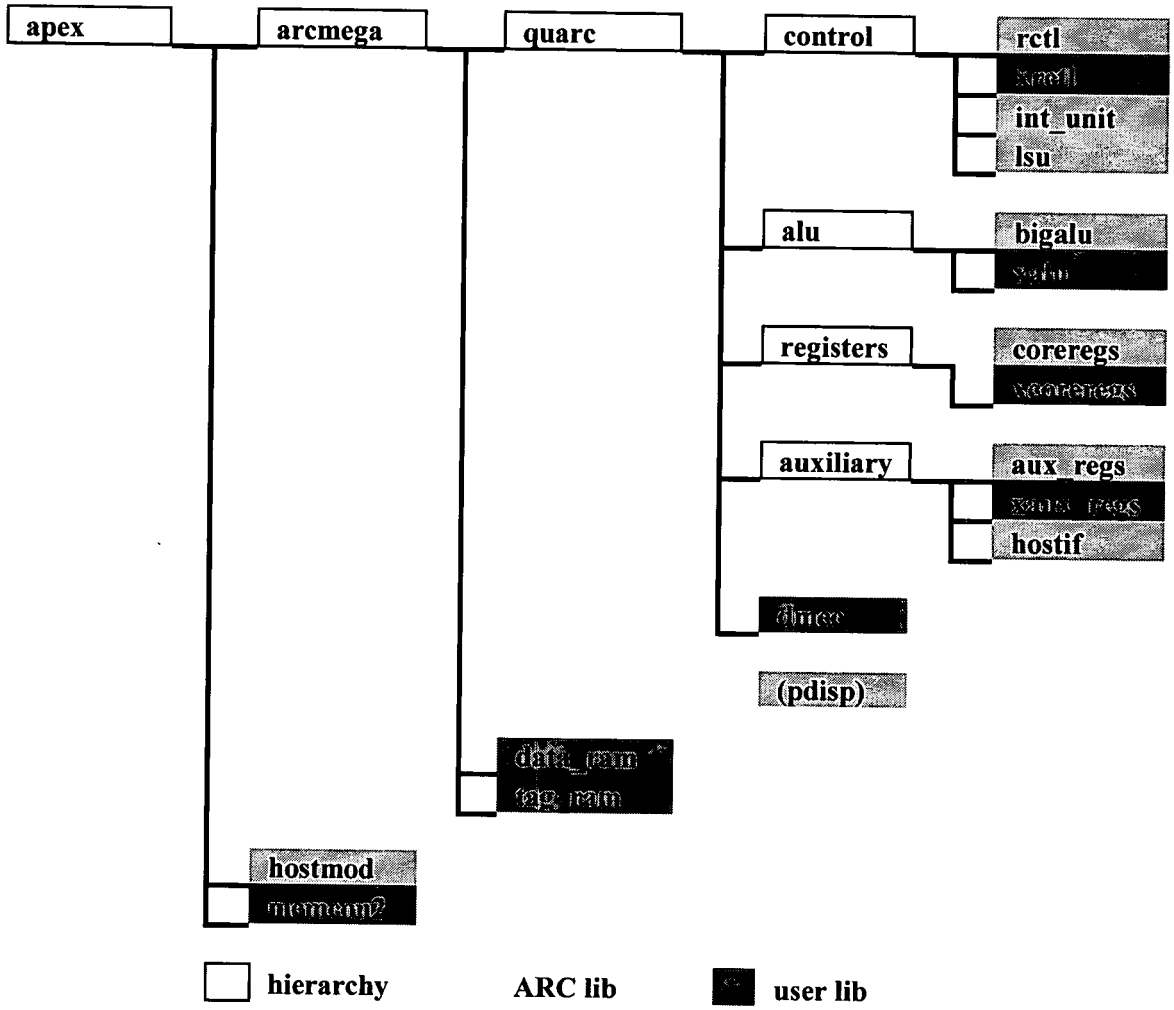
- Designed for safe customization of the core
- Basecase/Extension VHDL files separated
  - different directories
- ..but basecase and extensions synthesize together at the same hierarchical level.
  - basecase and extension pipeline control units can be synthesized as one block.
- easy addition of vendor-specific blocks
  - memories, megacells, IO pads, etc

VHDL libraries

- Basecase library - 'ARC'
  - basecase components
  - no editing

- User's library
  - instruction cache
  - other extensions
  - hierarchy
  - all editing takes place here

Design hierarchy



automatically generated hierarchy resides in user lib

Why generate the hierarchy?

- Extensions logic is split between different areas of the hierarchy :
- ALU, pipeline control, core regs, aux regs
- Interconnection required between user extension logic blocks
- Writing structural VHDL is very tedious

Hierarchy generation

- A big pool of design entities
  - in ARC and user's libraries
- Unique signal names must be used
  - so no netlist is required
- A control file defines the hierarchy
- The tool works out the routing of signals
- Hierarchy can be regenerated at any time
- Can override basecase ARC modules

ARC builders

- Build a customised ARC designs
- Specify cache configuration
- Select target clock speed and skew
- Builds extended ARC systems

Installation of VHDL release

- Extract the tape into a CAE area

In .cshrc:

- Set $ARCHOME environment variable
  - e.g. `setenv ARCHOME /apps/argonaut`
- Add bin directory to path -

```
set path = ($ARCHOME/system/bin $path)
```

- Set up other environment variables

```
setenv ARC_MANCODE <users manufacturer code>
setenv ARC_MANVER <users own version number>
setenv ARCASIC altera
```

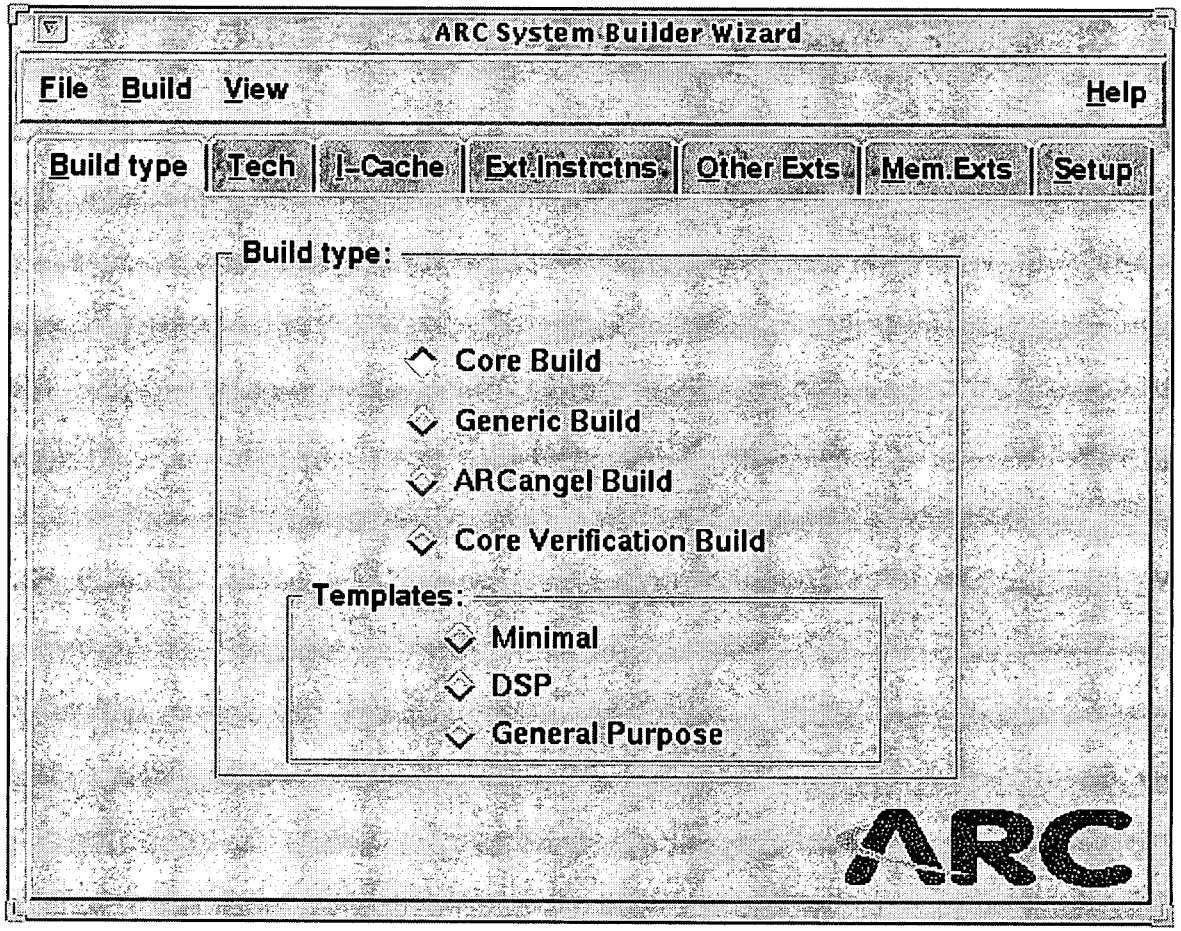
Document viewer

- Type 'arcdocs' to access document menu
- Requires Adobe Acrobat viewer
- HTML menu can be used
- ARC Programmer's Reference Manual
- ARC Extensions Interface Manual
- ARC Interface Manual
- Demonstrator board manual
- Training slides
- Builder user guides

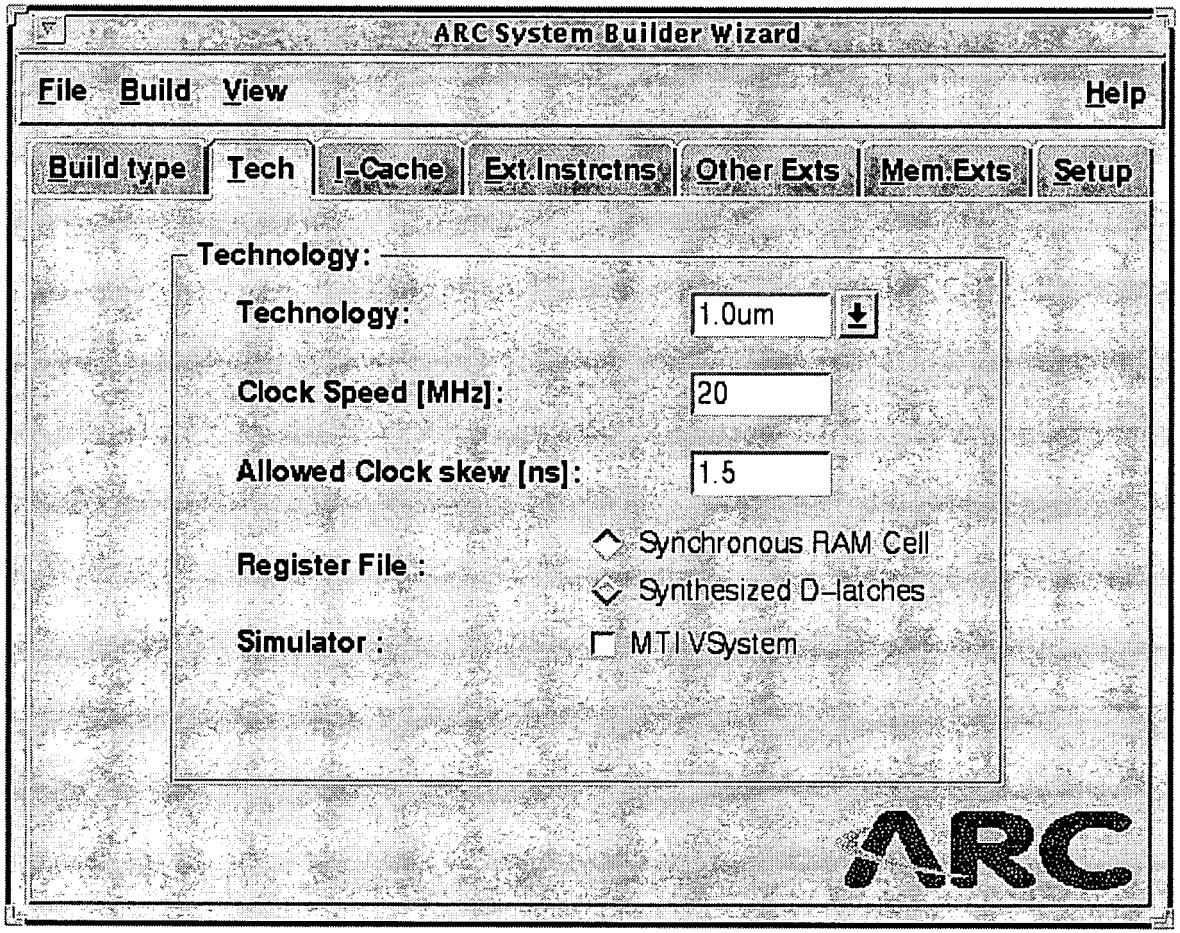
ARC Wizard

- GUI-based ARC system builder
- Uses TCL
- Type 'arc_wizard' to run ARC Wizard

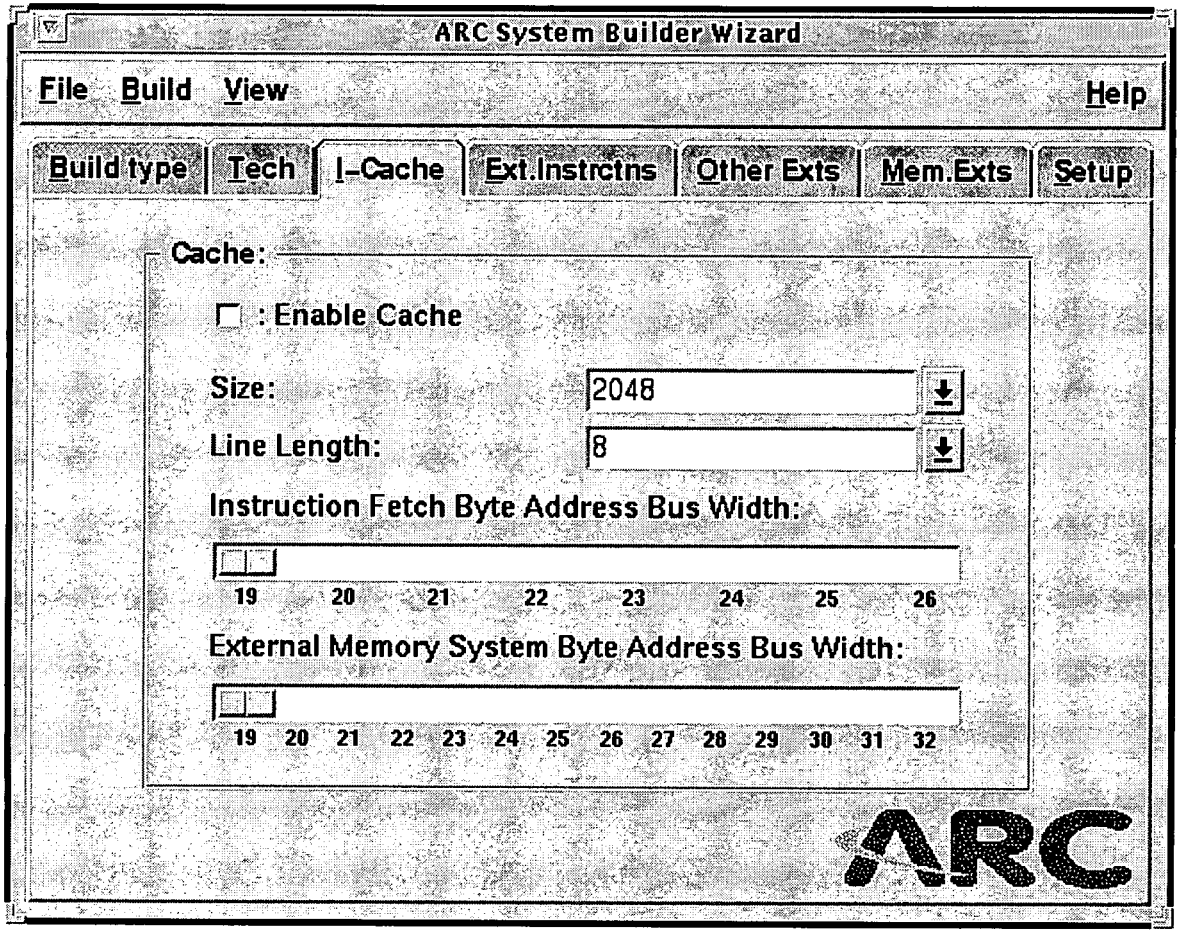
Build Type



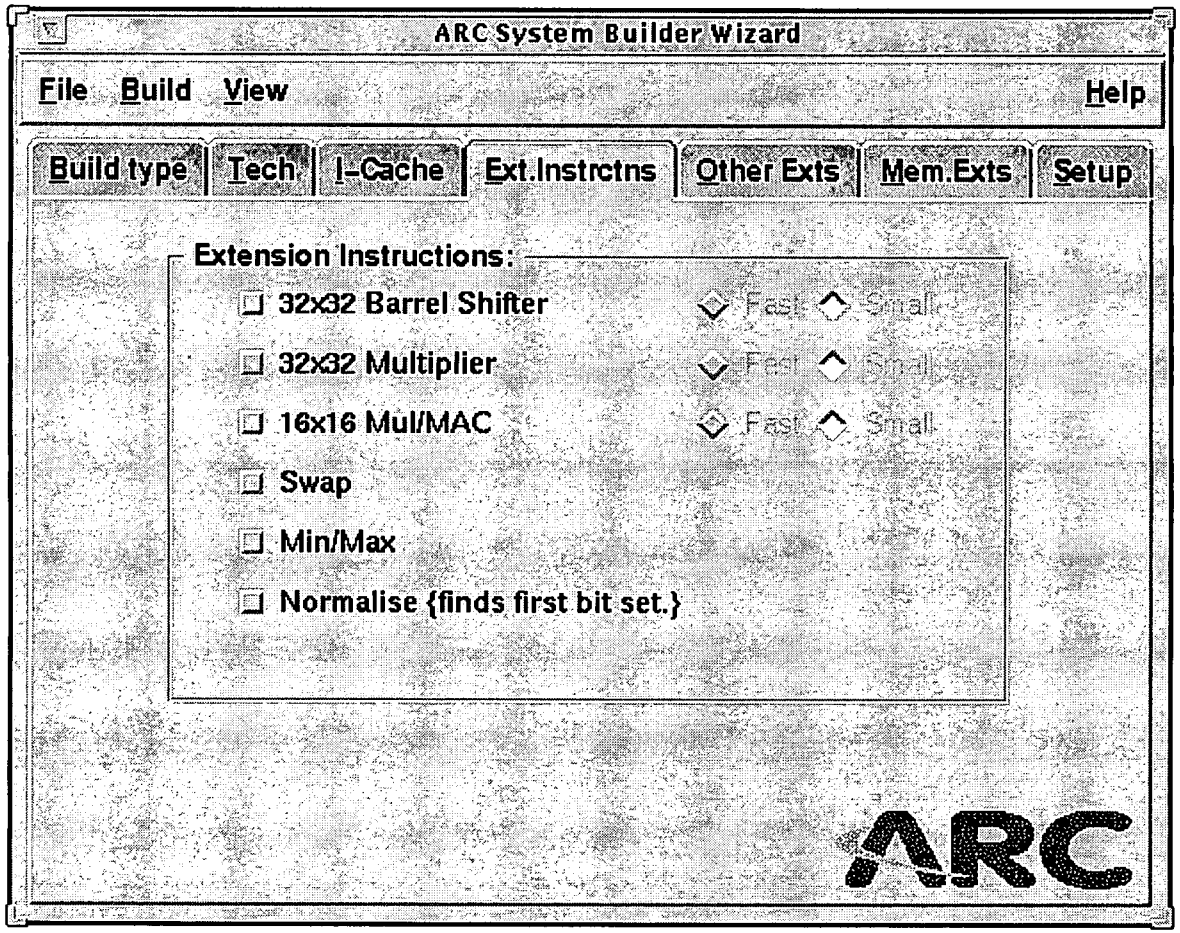
Technology



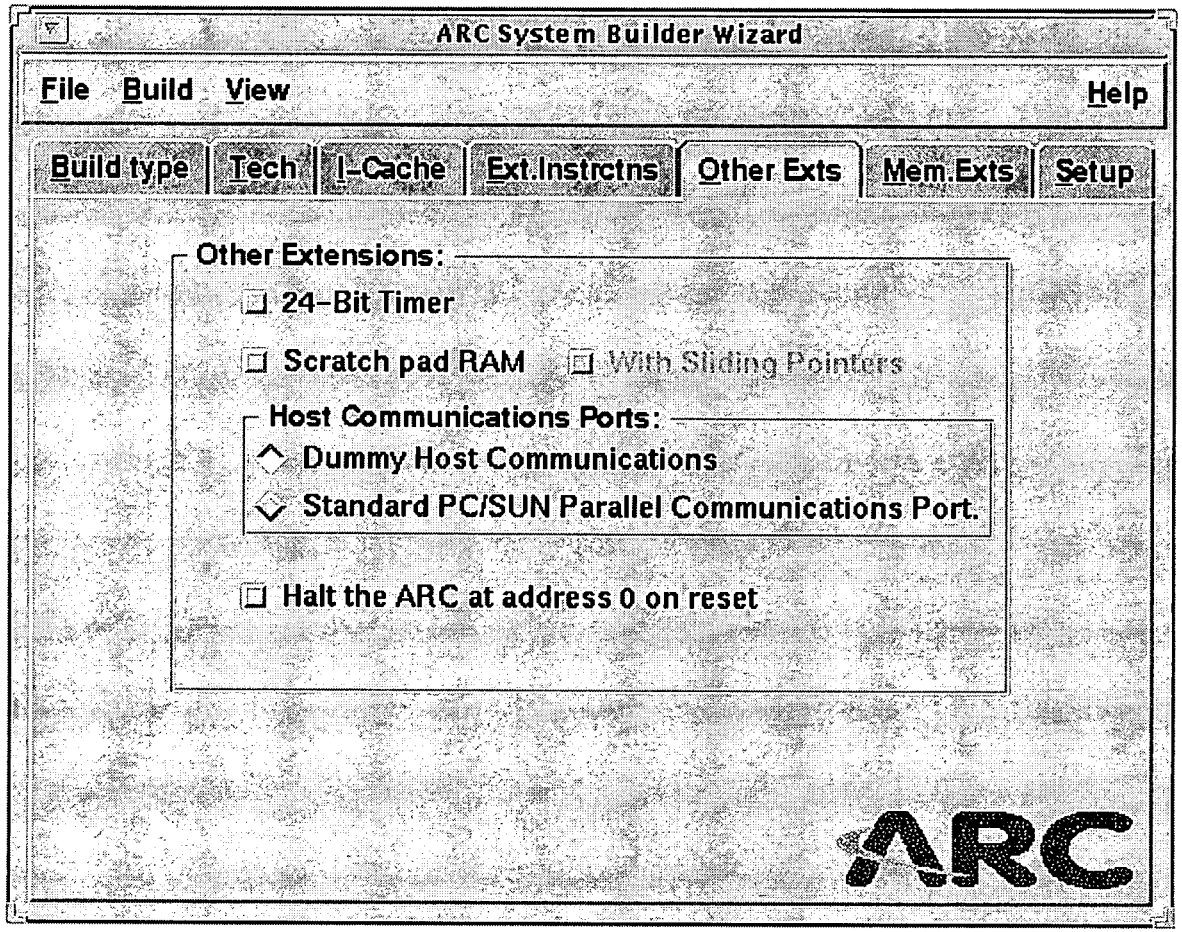
Instruction Cache



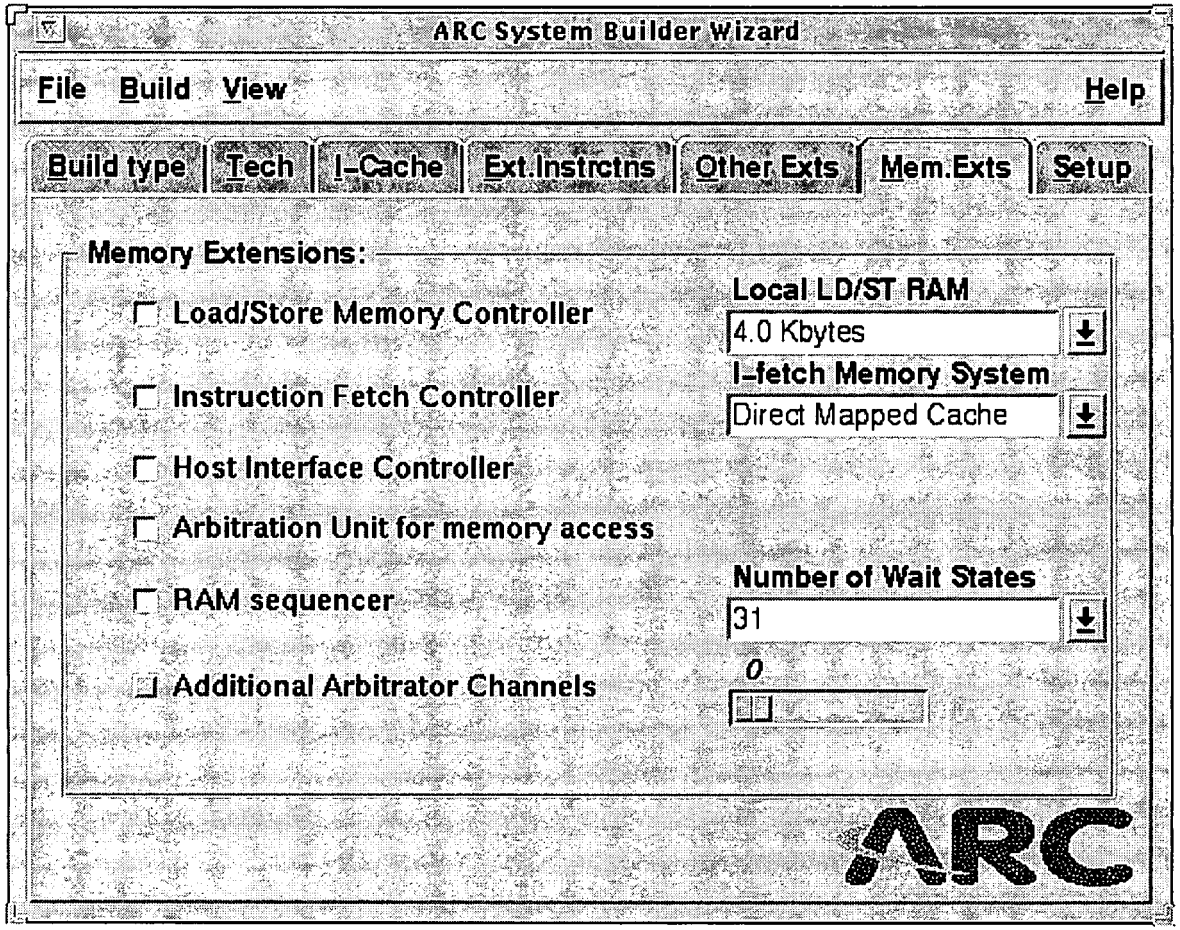
Extension Instructions



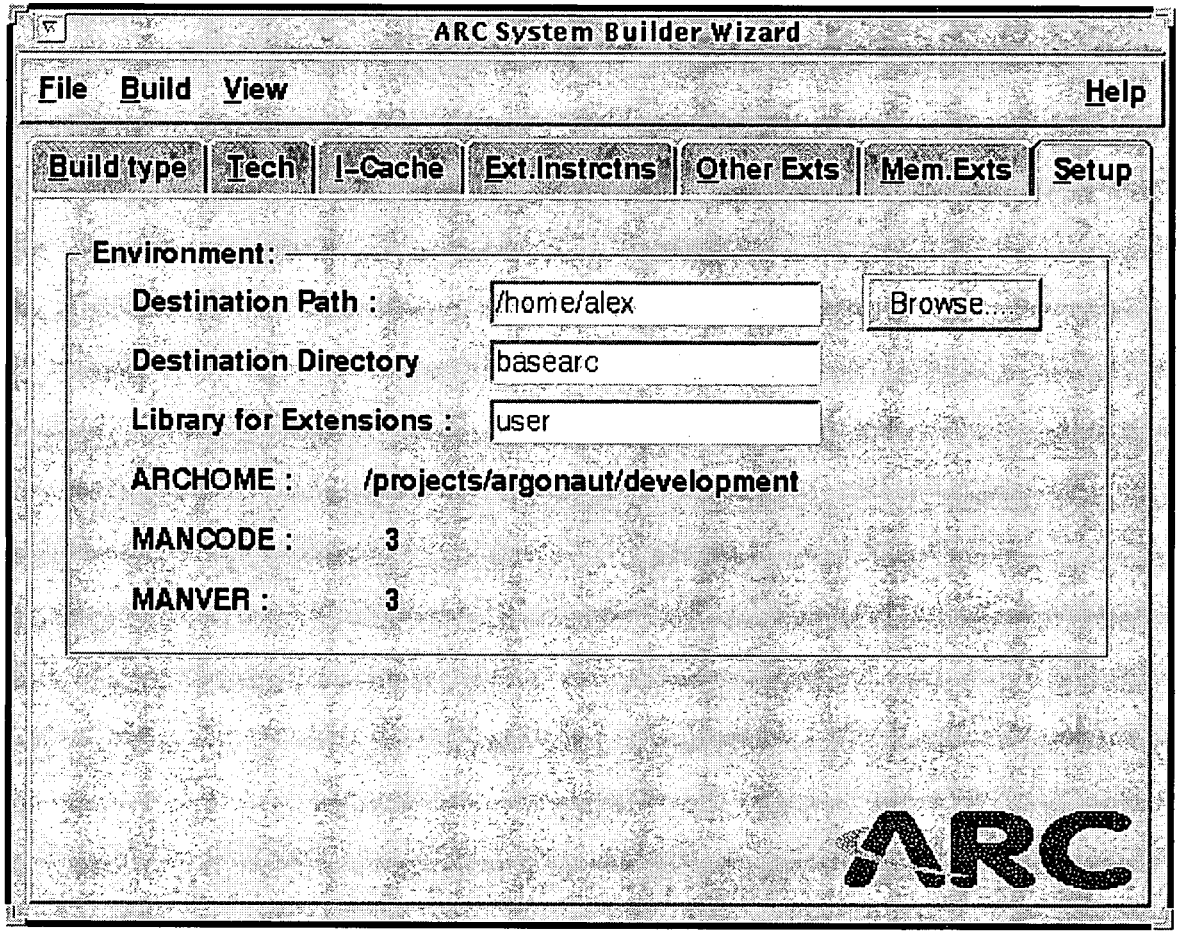
Other System Extensions



Memory System Extensions



Project Setup

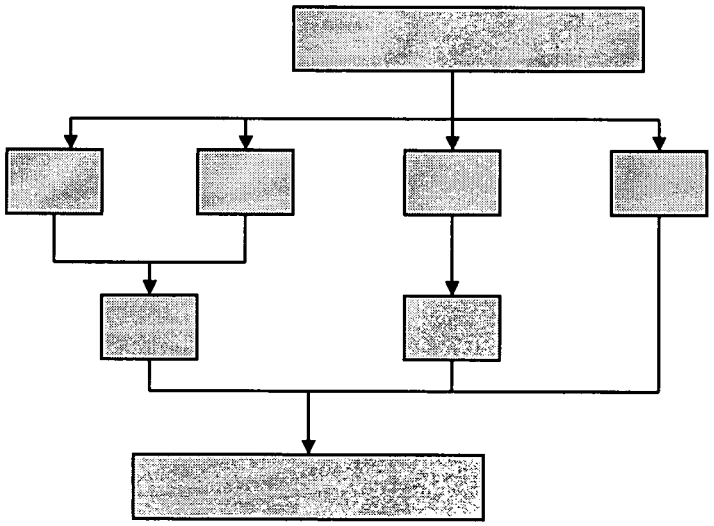


system_builder

- Command line based ARC system builder
- Type 'system_builder'
- Builds same systems as "ARC Wizard":
  - Build Type
  - Technology Details
  - I-Cache
  - Extension Instructions
  - System Extensions
  - Memory Extensions

System_builder

- User guide for system_builder:
  - System setup
  - Walk-through
  - How to run generated system
  - Full menu flow chart



Don't forget

- Narrow external memory bus (e.g. 16 bit) must emulate a 32-bit bus internally, to use existing cache logic.
- More external memory means more I-Cache tag bits.
- Full path to intall user's ARC
  - must not include /tmp_mnt/etc
  - a fresh directory is created
  - can not overwrite an existing directory
  - specify a new directory name for every install

In the working directory..

Directories for:

- VHDL
- Synopsys VHDL libraries
- Scripts
- Hierarchy generation system (/dat)
- Makefile (/anz)

In the working directory..

Files:

- Compilation makefile - for MTI VSystem, but easily modified.
- `user_synopsys_dc.setup` - to be included by `.synopsys_dc.setup`
- `init_mem.hex` - Datafile containing the basecase test code, loaded by the testbench.

ARC Simulation

Simulation

- Makefile for MTI Vsystem

Once VHDL is compiled...

- Run forever
  - simulation stopped by an `ASSERT`
- Testbench generates clocks and reset

Simulation

- A few 'X' warnings to begin with
- Reset (ARC halted)
- Simple host interface tests
- ARC started by host
- First cache line load (reset vector)
- Jump to the first piece of test code

Simulation

- Test code runs until completion
  - any errors show up as a premature stop
- When ARC halts, host checks to see if tests completed successfully and reports
- Simulation stopped with an assert-fail.
- Test code run again in single-step mode

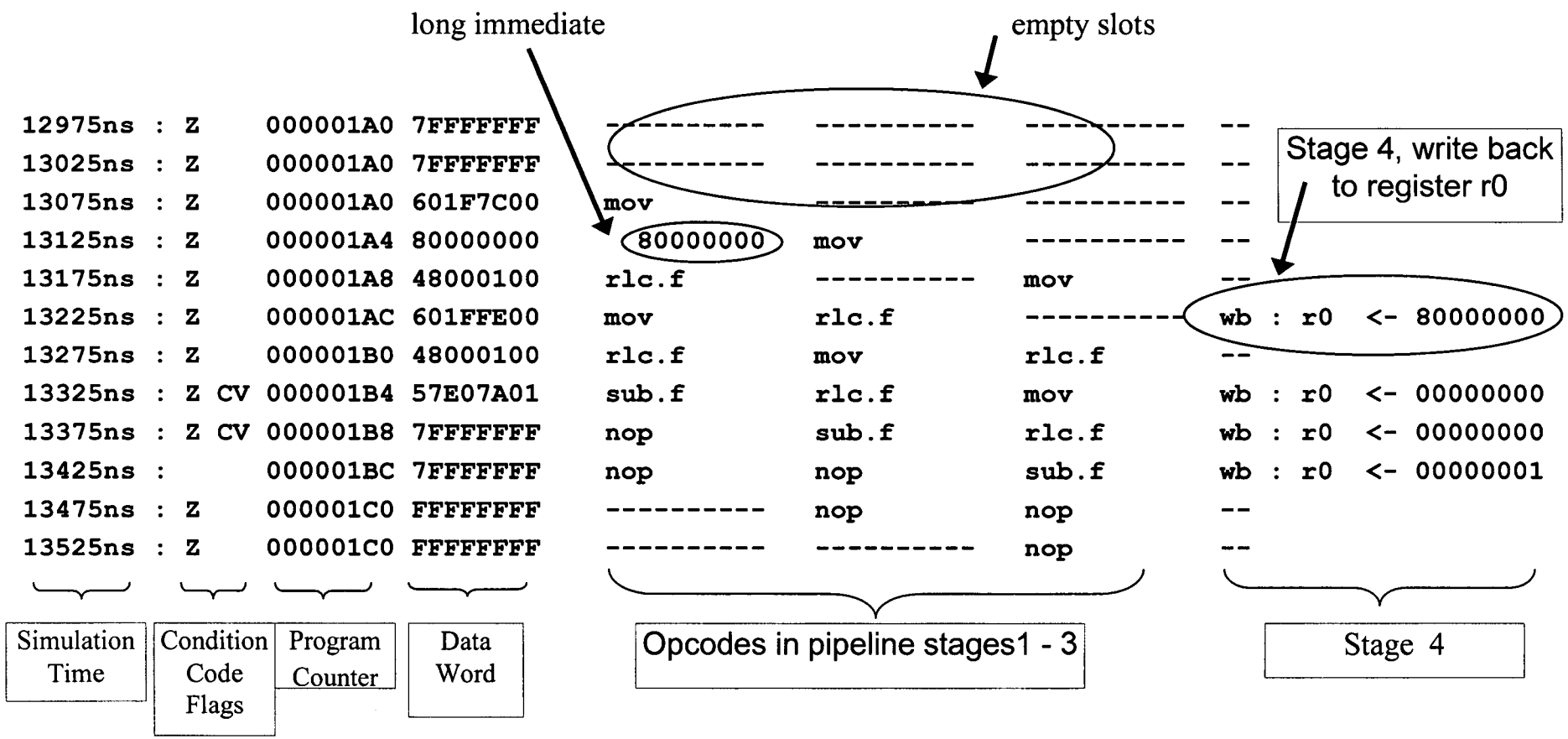
Test code

- Basecase ALU operations
- Register file, shortcut, writethrough
- Single instruction loops
- 32-bit memory ld/st, pipeline stalling
- 32x32->32 mul
  - First bit of code ever run on an ARC!
- flag setting by core instructions
- conditional instruction execution for core cc units
- byte-word-long loads and stores
- $ARCHOME/arc/src/metaware/coretest.s

Example of test code

```
; Stall pipeline using register interdependency

        mov     r2, testbase-4
        ld      r1, [r2, 4]     ; Load $FFFFFFFF into r1
        and     r4, r1, r6      ; Depends on r1 load => should stall
        or      r7, r8, r9
        sub     r10, r11, r12
        xor.f   r1, r1, $FFFFFFFF
        nop
        stop.nz 0
        xor.f   r4, r4, $F050A050
        nop
        stop.nz 0
        xor.f   r7, r7, $FFFFFFFF
        nop
        stop.nz 0
        sub.f   r10, r10, 127000
        nop
        stop.nz 0
```

Simulation display



Simulation versus Disassembly

```
27225ns : Z    00000368 613F7C00   mov          flag         ----------  --
27275ns : Z    0000036C 80000002     80000002   mov          flag        --
27325ns :      00000370 49249300   rlc.f        ----------   mov         --
27375ns :      00000374 49249300   rlc.f        rlc.f        ----------  wb : r9  <- 80000002
27425ns :      00000378 57E4FA09   sub.f        rlc.f        rlc.f       --
27475ns :   CV 0000037C 7FFFFFFF   nop          sub.f        rlc.f       wb : r9  <- 00000004
27525ns :      00000380 FFFFFFFF   ----------   nop          sub.f       wb : r9  <- 00000009
27575ns : Z    00000380 FFFFFFFF   ----------   ----------   nop         --
27625ns : Z    00000380 FFFFFFFF   ----------   ----------   ----------  --
*************** No State Change for 11 pdisp updates ***************
28225ns : Z    00000380 7FFFFFFF   nop          ----------   ----------  --
28275ns : Z    00000384 20000082   bne          nop          ----------  --
28325ns : Z    00000388 20000200   b            bne          nop         --
28375ns : Z    0000038C 1FBF0002   flag.ne      b            bne         --
```

```
    368:  613f7c00  80000002  mov          %r9,0x8000_0002
    370:  49249300            rlc.f        %r9,%r9
    374:  49249300            rlc.f        %r9,%r9
    378:  57e4fa09            sub.f        0,%r9,9
    37c:  7fffffff            nop
    380:  7fffffff            nop
    384:  20000082            bne          0x38c = _start+0x28c
    388:  20000200            b            0x39c = _start+0x29c
    38c:  1fbf0002  00000000  flag.ne      0
```

Simulation log notes

- [...] - ARC Stopped
- (...) - This stage is stalled
- ---- - Empty slot
- Delay slot cancelling modes reported
- Loads and stores reported
- Host accesses reported
- Extensions instructions can be added to the simulator

Simulation Display

```
8075ns :      00000110 605F7C00   mov          ----------   ----------  --
8125ns :      00000114 DECEA5ED     DECEA5ED   mov          ----------  --
8175ns :      00000118 7FFFFFFF   nop          ----------   mov         --
8225ns :      0000011C 7FFFFFFF   nop          nop          ----------  wb : r2  <- DECEA5ED
8275ns :      00000120 FFFFFFFF   ----------   nop          nop         --
8325ns :      00000120 FFFFFFFF   ----------   ----------   nop         --
8375ns :      00000120 FFFFFFFF   ----------   ----------   ----------  --
8425ns :      00000120 FFFFFFFF   ----------   ----------   ----------  --
8475ns :      00000120 FFFFFFFF   ----------   ----------   ----------  --
8525ns :      00000120 FFFFFFFF   ----------   ----------   ----------  --
8575ns :      00000120 11000204   ----------   ----------   ----------  --
8625ns :      00000120 11000404   ----------   ----------   ----------  --
8675ns :      00000120 08800000   ----------   ----------   ----------  --
8725ns :      00000120 086001FC   ----------   ----------   ----------  --
8775ns :      00000120 57E08700   ----------   ----------   ----------  --
8825ns :      00000120 7FFFFFFF   ----------   ----------   ----------  --
8875ns :      00000120 20000082   ----------   ----------   ----------  --
8925ns :      00000120 20000200   ----------   ----------   ----------  --
8975ns :      00000120 11000204   st.a         ----------   ----------  --
9025ns :      00000124 11000404   st.a         st.a         ----------  --
9075ns : >>> ARC Store A=00004E30 D=CAFEF00D longword  <<<
9075ns :      00000128 08800000   ld           st.a         st.a        --
9125ns : >>> ARC Store A=00004E34 D=DECEA5ED longword  <<<
9125ns :      0000012C 086001FC   ld           ld           st.a        wb : r0  <- 00004E30
9175ns : >>> ARC Load  A=00004E34 longword  <<<
```

Showing state changes only

```
 7875ns :      00000100 601F7C00   mov          ----------   ----------  --
 7925ns :      00000104 00004E2C     00004E2C   mov          ----------  --
 7975ns :      00000108 603F7C00   mov          ----------   mov         --
 8025ns :      0000010C CAFEF00D     CAFEF00D   mov          ----------  wb : r0  <- 00004E2C
 8075ns :      00000110 605F7C00   mov          ----------   mov         --
 8125ns :      00000114 DECEA5ED     DECEA5ED   mov          ----------  wb : r1  <- CAFEF00D
 8175ns :      00000118 7FFFFFFF   nop          ----------   mov         --
 8225ns :      0000011C 7FFFFFFF   nop          nop          ----------  wb : r2  <- DECEA5ED
 8275ns :      00000120 FFFFFFFF   ----------   nop          nop         --
 8325ns :      00000120 FFFFFFFF   ----------   ----------   nop         --
 8375ns :      00000120 FFFFFFFF   ----------   ----------   ----------  --
*************** No State Change for 11 pdisp updates ***************
 8975ns :      00000120 11000204   st.a         ----------   ----------  --
 9025ns :      00000124 11000404   st.a         st.a         ----------  --
 9075ns : >>> ARC Store A=00004E30 D=CAFEF00D longword  <<<
 9075ns :      00000128 08800000   ld           st.a         st.a        --
 9125ns : >>> ARC Store A=00004E34 D=DECEA5ED longword  <<<
 9125ns :      0000012C 086001FC   ld           ld           st.a        wb : r0  <- 00004E30
 9175ns : >>> ARC Load  A=00004E34 longword  <<<
 9175ns :      00000130 57E08700   sub.f        ld           ld          wb : r0  <- 00004E34
 9225ns :      00000134 7FFFFFFF  (nop       ) (sub.f     ) (ld        ) --
 9275ns : >>> ARC Load  A=00004E30 longword  <<<
 9275ns :      00000134 7FFFFFFF  (nop       ) (sub.f     )  ld         ld : r4  <- DECEA5ED
 9325ns : >>> Data DECEA5ED coming back from load <<<
```

Running code

- Object code converted to hex format readable by VHDL testbench
- Metaware tools available for Sun and PC
- Symbolic link to PC directories
  - where PCNFS is available.

ARC Synthesis

Synthesis

- VHDL optimised for Synopsys Design Compiler Expert
- Use with other Synthesis tools may require some modifications.
- Synthesize to gate array, standard cell, FPGA.

Technology Independence

- Pipeline is fully static RTL-VHDL
- Generic RAM blocks are replaced by Vendor-supplied models.
- Top-down synthesis methodology
  - Set up libraries and clock speed
  - Run synthesis script
  - Optimize if necessary

ASIC Vendor

- RAM models for VHDL and Synopsys
- Macrocells
- Clock tree synthesis
- Scan insertion
- ATPG
- Floorplanning tools + SDF generation
- Layout

Register file

- One write, two reads simultaneously

Either..

- Synthesize from d-latches
- Use asynchronous RAM cell
- Use synchronous RAM cell
- Muxers can be used for writethrough

Synthesis scripts

- `./scripts/analyze.dc`
  - analyzes all VHDL for ARC and user libs
- `./scripts/elaborate.dc`
  - elaborates and flattens hierarchy
- `./scripts/do_all.dc`
  - constraints, compile, and optimisation
- `syn_arc`
  - constraints, compile, and optimisation

Analyze script

scripts/analyze.dc

- Compiles basecase and user VHDL
- Modify if hierarchy is altered.

Elaborate script

scripts/elaborate.dc

- Elaborates basecase and user VHDL
- Flattens logic to the five basic levels :
  - ALU
  - RISC control
  - Instruction cache
  - Core registers
  - Auxiliary registers
- Modify if hierarchy is altered.

Master script

- Calls analyze, elaborate scripts
- Sets up
  - input/output delays and loads
  - clock constraints
  - some tweak variables
- first compilation - may take some time
- incremental compiles on adder blocks
- global incremental compile
- timing and area reports

Running synthesis

- Type 'syn_arc'

  => dc_shell -f scripts/do_all.dc

- Or better,

```
syn_arc >! syn_arc.log &
tail -f syn_arc.log
```

Synthesis run times

- Dependent upon technology and constraints
- Gate array typically faster than standard cell
- Sun SparcStation 20 (HyperSparc)
  - Basecase ARC, synthesize register file
  - 0.35um GA, to 85MHz - 6h30m
  - 0.35um GA, 50 Mhz - 2h20m

Typical synthesis problems

- Adders - These often require extra compilation to improve speed.
- Setup/hold times on cache memory blocks
- Poorly coded extension VHDL
  - careless use of pipeline enable signals

Synthesis problems

Common critical paths:

- Cache tag output, instruction valid, new PC value, cache RAM address inputs.
- Via ALU adder block (32+32 add/sub)
- Register shortcut path to new PC value, to cache RAMs or PC flip-flops.
  - e.g.

```
add     r0,r0,r0
jal     [r0]
```

ARC Development Board

What is it?

- Reconfigurable hardware
- A prototyping system for ARC extensions
- Development platform for ARC software

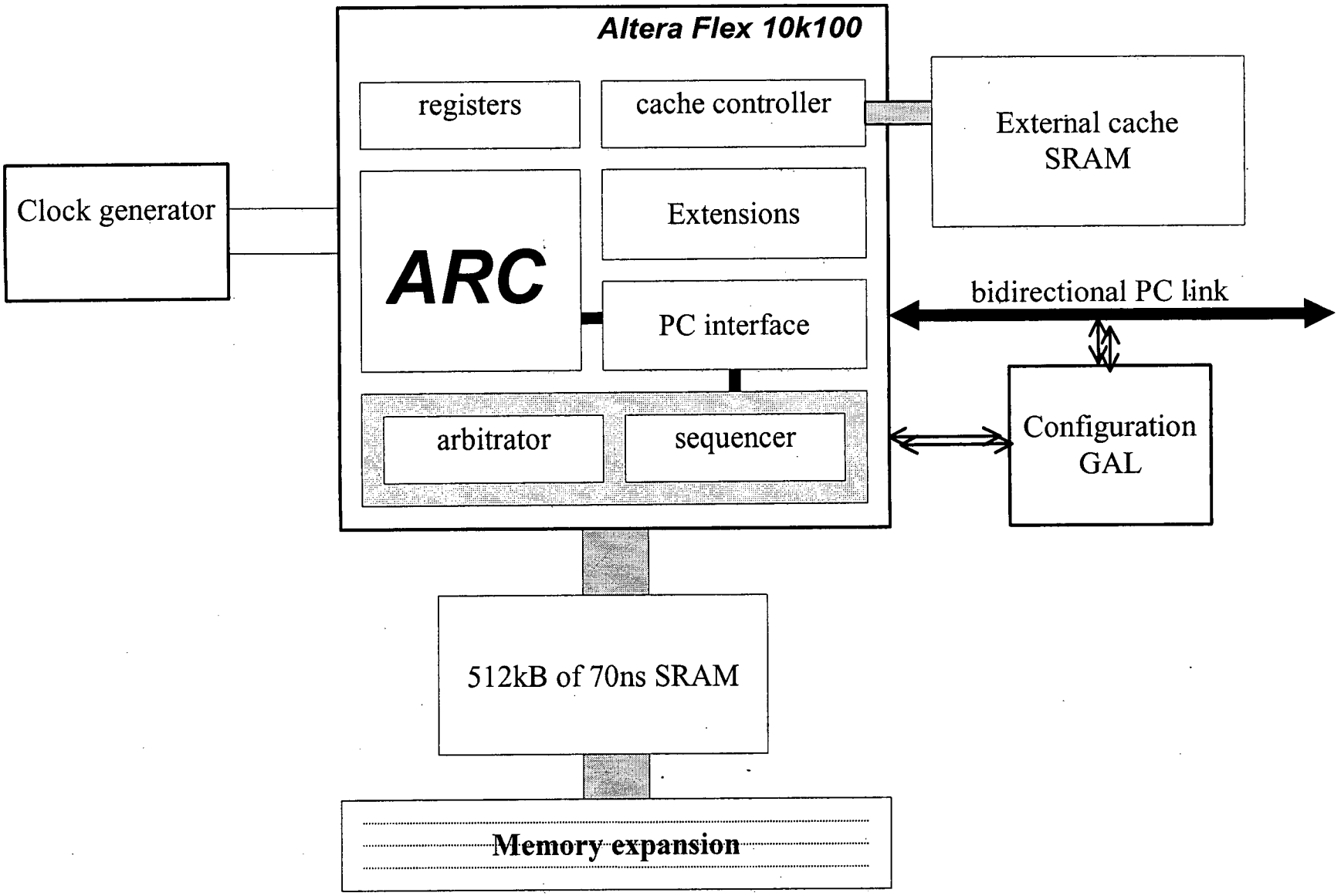
- A blank canvas for ARC development.

Why is it useful?

- Basic system
  - Code development and debugging
  - Cache size tuning
- Extension prototyping
  - software/hardware codesign
  - optimisation of inner loops
  - fast design iterations

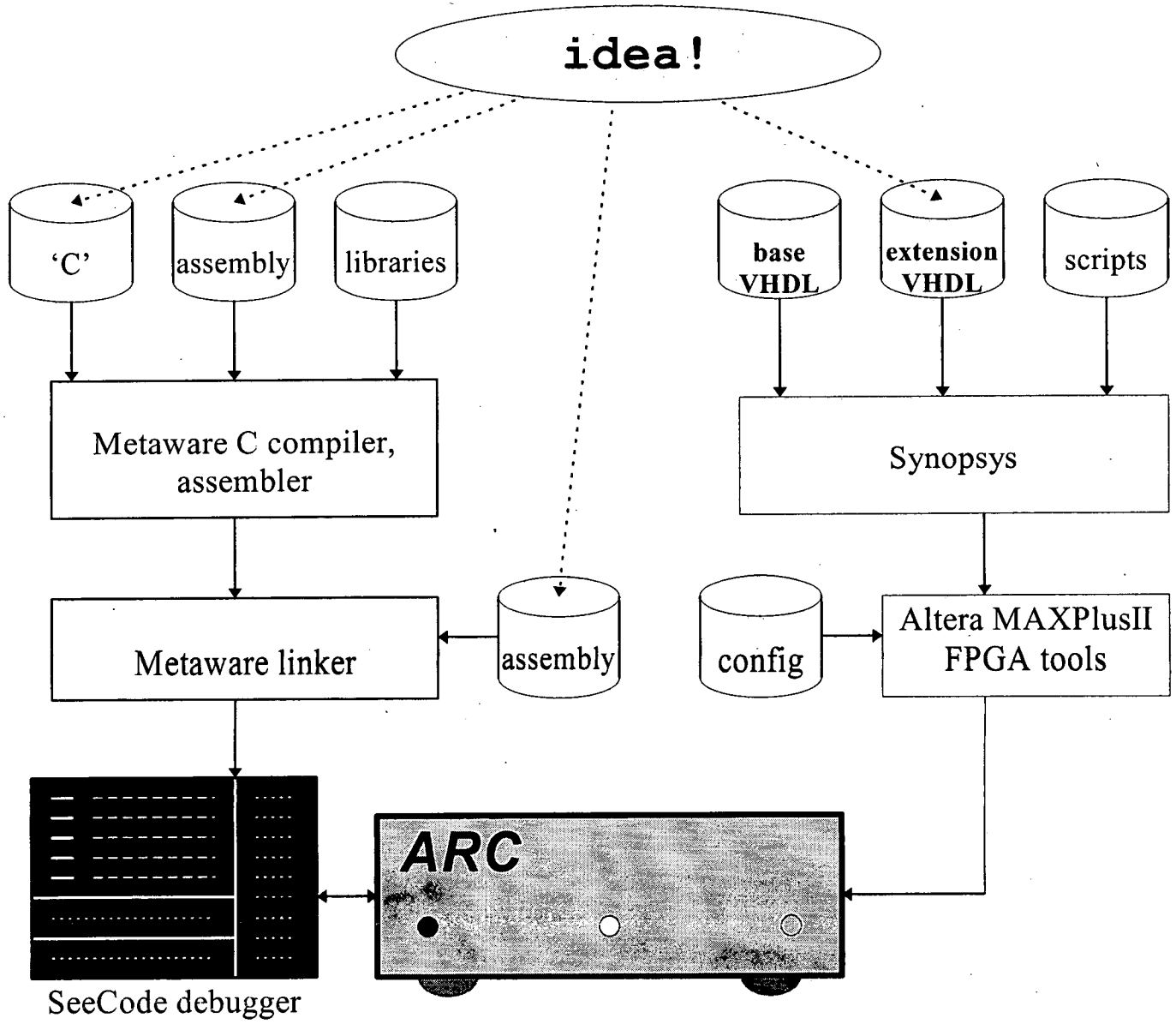
Resources

- Based around Altera 10k100 FPGA
- Basic design uses
  - 57% of logic resources
  - 4% of on-board RAM resources
- 512Kb main memory for code/data
- Select i-cache size from 0-16Kb
- Bidirectional PC interface
- Cycle timer

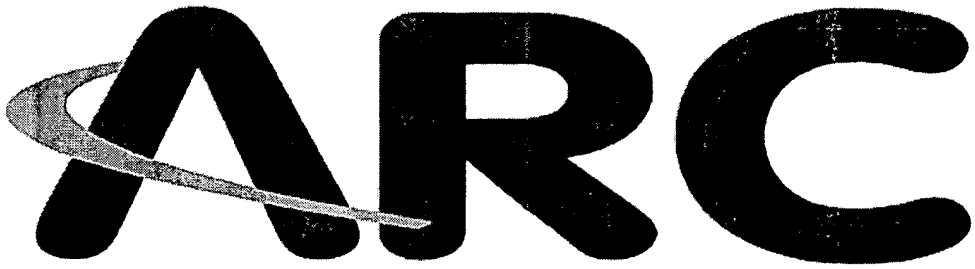


Design flow for prototyping

- Similar to ASIC flow
- Synopsys DC used with Altera libraries
- Altera P+R tools required
- Some Altera-specific modules
  - but extensions VHDL stays the same
- Quick design iterations
  - VHDL to download from 3 hours



Questions



ARC Configuration Wizard User Guide

Document revision - v 1.5

Author:	Alexander Holland
Last revised by:	Alexander Holland
Revision:	1.5
Date of last revision:	14 July 1998
Filename:	wizard.pdf

Contents

Table of Figures

Preface

This document describes the Graphical User Interface which complements the ARC System builder, the build script used for creating files for simulation on a VHDL simulator and VHDL synthesis.

This document is aimed at ASIC engineers working with the ARC.

1 What is the ARC wizard?

The ARC wizard is a 'graphical front end' to the system builder script, for Sun and Solaris, that allows users to build customised ARC systems along with support files for simulation and synthesis. The wizard replaces the series of questions the build script produces (See System builder script user guide) and replaces it with, a 'point and click' environment. The settings chosen using the wizard are then passed to the system builder script which builds the VHDL simulator and synthesis files. The wizard has all the functionality of the text script, with the extra facility of being able to save configurations and ***'by its very nature' the ability to change settings at any time before a build.*** It's exclusion system prevents the user from selecting settings that will lead to a non-functioning ARC, while leaving the maximum flexibility for customisation

- The user can select from 4 different types of build:
  - ◊ Core Build
  - ◊ Generic System Build
  - ◊ Altera Build (for ARCAngel development board)
  - ◊ Core Verification Build (for ARC Core verification tests.)

- The wizard can be used to select all the following ARC features :
  - ◊ Fast Load returns
  - ◊ Extension Instructions required
  - ◊ Memory Extensions required
  - ◊ Cache Mode
    - ◊ Cache size
    - ◊ Cache line length
  - ◊ Size of external memory space that is to be cached.
  - ◊ Clock period
  - ◊ Clock skew
  - ◊ Synthesised d-latches or 3-port RAM for register file
  - ◊ RAM model timings
  - ◊ RTL and SEECODE Application Link (RASCAL)
  - ◊ Halt the ARC on reset
  - ◊ Manufacturer code and version number information

When complete a users selections are passed to the script as parameters.

- The script creates a working directory for the user with in which it creates the following:
  - ◊ VHDL for the extensions selected
  - ◊ VHDL for a direct mapped I-cache configured to user's specification
  - ◊ VHDL test-bench for testing basecase operation and external interfaces
  - ◊ VHDL structure to link together all required modules
  - ◊ Configuration files for Synopsys Design Compiler
  - ◊ Memory image file containing basecase test code
  - ◊ Synthesis script for Synopsys Design Compiler v3.4b or above
  - ◊ Makefile - set up for the Model Technologies VSystem/VHDL simulator, but can be altered for use with other simulation environments.

2. Setting up the wizard

2.1 Setup

Before using the ARC wizard, several environment variables have to be set.
In .cshrc:
Set the **ARCHOME** environment variable to the base of the ARC install tree. Add the bin directories to the path. Set the environment variables such as **ARC_MANCODE**, **ARC_MANVER** and **ARCASIC**. (If these variables are not set, the build script will ask for them at run time.)

E.g.

```
setenv ARCHOME /apps/argonaut
setenv ARC_MANCODE <Users Manufacturer code>
setenv ARC_MANVER <Users selected version number>
setenv ARCASIC <Build type>

set path = ($ARCHOME/arc/bin)
```

The **SYNOPSYS** and **MAX2_HOME** variables must also be set. If you have access to *Synopsys* and the ALTERA FPGA tool *Max plus II,* from your shell, these variables should already be set.

ARC_MANCODE is the unique manufacturer code that Argonaut gives to each ARC licensee, and the **ARC_MANVER** is that code that allows manufacturers to have their own version number. Both numbers go on to be stored as part of the identity register. (For more information on the identity register, see the ARC Programmers Reference Manual.)

The **ARCASIC** environment variable is used to automatically choose the build type in the text version of the script. It should be set to either, **altera, generic, core, core_verification** or left not set. If set to nothing, the text version will prompt the user for a build type. The wizard does NOT use the **ARCASIC** environment variable.

The **SYNOPSYS** variable is the location of the Synopsys VHDL synthesis tool.

The **MAX2_HOME** is the location of the *MAX plus II* EDA tool for use with the Altera Flex10K 100 FPGA used in ARCAngel boards.

2.12 Setting up Tcl/Tk and Tix

For the wizard to work you must have both Tcl/Tk, 8.0p2 or later and the Tix shared object library installed on your system. These are available from http://sunscript.sun.com/ and http://www.xpi.com/tix/index.html and should be installed by the network administrator. NOTE: There must be a path to the TCL `bin` directory in the shell that you are launching the wizard. If you have any difficulty in setting up these tools, the ARC wizard can be supplied as a binary executable for SUNOS 4.x (Solaris 1.x) and SUNOS 5.x (Solaris 2.x).

2.13 Launching the ARC wizard.

After setting up the environment, the ARC System Builder wizard can be launched by typing **arc_wizard**. After a few seconds the window will appear.

3. Using the ARC Wizard

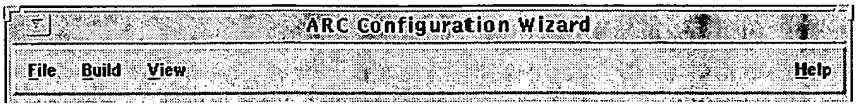


Figure 1: Menu Bar

3.1 Menu Bar

The menu bar at the top of the ARC wizard has four categories,

Clicking on the menu bar under **FILE** or holding down **ALT+F** will display the file pull down menu. From here, the user can use the mouse or arrow keys to select an action.

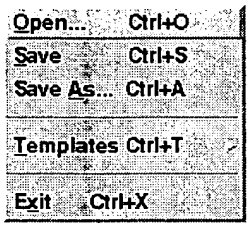


Figure 2: File pull down menu

Open... Will produce a file requester dialogue box set to the current directory. The file extension for ARC wizard saves is `.cfg` and so the dialogue box defaults to this file type, the user can however look for, open and save files with any extension. At any time, the user can call the *open* dialogue box by pressing **Ctrl+O**.

Save Automatically saves the configuration under the active filename. If there is no active filename (for instance if the user has not opened a file) the wizard defaults to the '`buildtype.cfg`'. The user can save at any time by pressing **Ctrl+S**.

Save As... Creates a 'Save as' file requester dialogue box set to the current directory. NOTE: The user must enter a file name WITH file type extension.

Templates This is a cascade entry. Selecting this option will display a list of the templates currently available. A template is just an ordinary saved configuration containing general settings pertaining to a particular style of ARC. The ARC System builder

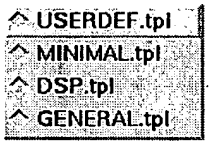


Figure 3: Template cascade

wizard comes with three default templates, **Minimal**, **DSP** and **General**. If a user regularly builds for example, the same technology and cache set-ups, they may wish to create a user-defined template. A user can define a template by saving a configuration with file type `.tpl` in the **$ARCHOME/wizard/templates** directory. Next time they run the wizard the new template will be amongst the list.

Exit To quit the ARC wizard either press **Ctrl+X** or select **Exit** from the file pull down menu, this will bring up a confirmation dialogue box from which the user can quit the wizard.

Pressing **ALT+B** displays the other pull down menu with one entry, **Build**. This is the same as clicking on the menu bar -> build, or clicking on the spanner in the tool bar. Selecting this brings up a log window and begins the build process.

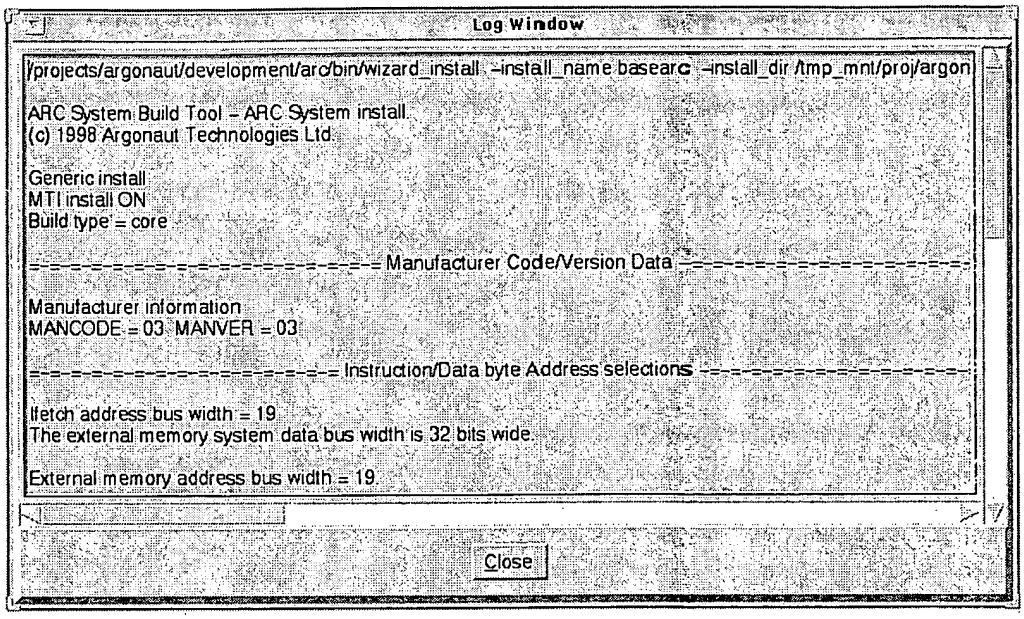


Figure 4: Log Window

Displayed in the log window (figure 4) are messages as to how the build is proceeding. While the log window is open the wizard is disabled. If for any reason you wish to stop mid build you can do so by pressing close, if directories have been created the user will be prompted if they want to clean up, i.e. remove all created files. All the information displayed in the window is saved in a log file (`sys_bld.log`) created in the destination directory.

The **View** menu bar option, allows the user to display a build summary (for more information see section 2.28).

Help menu bar option gives the user two further choices, View this documentation via Acrobat Reader, or see the **About** dialogue box.

3.2 Tool and Status Bars

The tool bar is a quick and easy way of launching some of the ARC wizards features. It consists of six iconised buttons stacked vertically on the right hand side of the wizard. From top down, these buttons are **Open**, **Save**, **Balloons ON/OFF**, **Programmers Model**, **Summary** and **Build**. The functions of five are described elsewhere in this document.
Balloons ON/OFF toggles the pop-up balloons that appear informing the user of a features individual gate count, and the status bar at the bottom of the page. By default they are off.

All the features of the ARC wizard have a small description of their purpose. Positioning the mouse over a particular feature causes the description to be displayed in the status bar at the bottom of the ARC wizard.

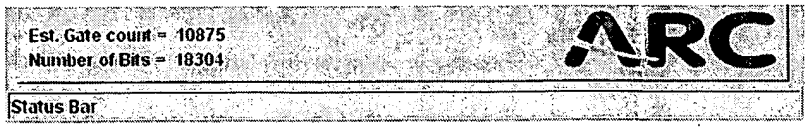


Figure 5: Status Bar, Estimated Gate and Bit counts.

Also at the bottom of the wizard is an estimate of the gate count and bit count for the current design. As ARC extension instructions, memory extensions etc., are added to / removed from the design the gate count is updated.

3.3 Build Exceptions

There are several reasons why the build may halt before completion, each of these display a descriptive error in the log window

1. MANCODE and/or MANVER not set.
2. A directory or file already exists with same name as Destination Directory that is to be created.
3. The User does not have write access to the destination path.
4. Files required for a build do not exist

If a license for ModelTech VSystem, MAX Plus II or Synopsys is not available at the time of building, a message will be displayed in the log window and the build will continue.

If a user is using MTI VSystem, and it is not available during a build, the VHDL will not be compiled. Allow the build to finish, then wait until a licence is available; enter the build directory, type **make** and the VHDL files will be compiled.

When using the System builder to build an ARCAngel, if a MAX Plus II licence is not acquired during a build, the RAM's are not generated. Again, allow the build to finish, wait for a license to become available and then use the genmem facility as explained in the ARC Development board (ARCAngel) document.

A Synopsys licence not being available at build time results in the Synopsys RAM models not being built. Depending on what RAM selections have been, different RAM generation scripts have to be run from the build directory.

Direct Mapped Cache RAM / Synchronous Register file
`make_arc_rams .`

Scratch Pad RAM / Scratch Pad RAM
`make_scratch_rams .`

Local LD/ST RAM
`make_local_rams .`

Altera TAG RAM
`make_tag_ram .`

Note: The first parameter for each of these commands (in this case '.') is the location of the working directory.

3.4 Build type Page

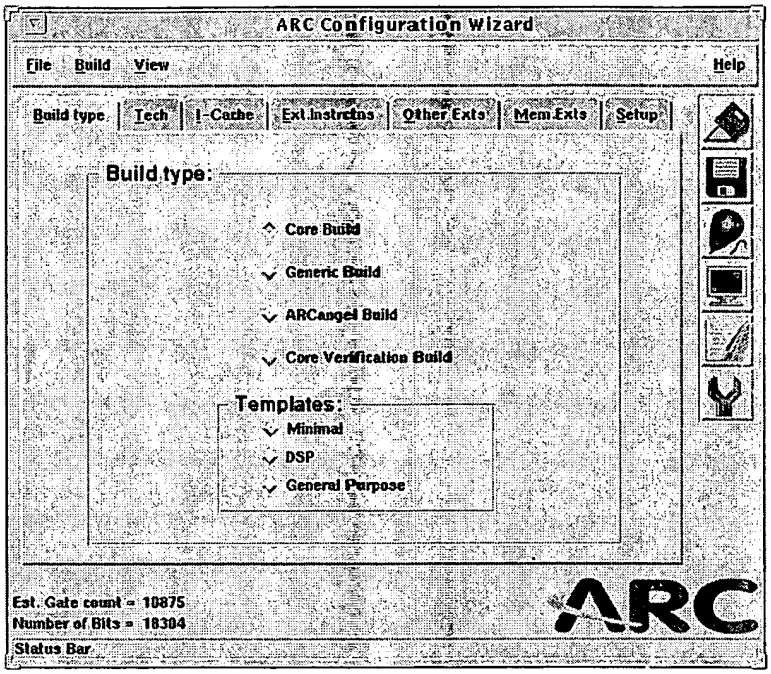


Figure 6: Build type Page

The build page is the first page encountered by the user. It allows a user to select from the four types of build available

- Core - A very simple ARC with just the core registers and any selected extension instructions, plus an optional instruction cache.
- Generic - A customised ARC with extension instructions, memory extensions and an instruction cache.
- ARCAngel - An ARC build for the ARCAngel development board.
- Core Verification - A build to allow the core-verification programs to be run.

Selecting one of the above enables and disables certain features of the Wizard unique to a particular build.

Below the Build types are three templates, clicking on one of these buttons will load a template, containing settings for a particular style of ARC. It is possible to create custom templates and store them in the template directory e.g. `$ARCHOME/wizard/templates`
A user can access these new templates via the menu bar at the top of the wizard. e.g. `File -> Templates.`

3.5 Technology Page

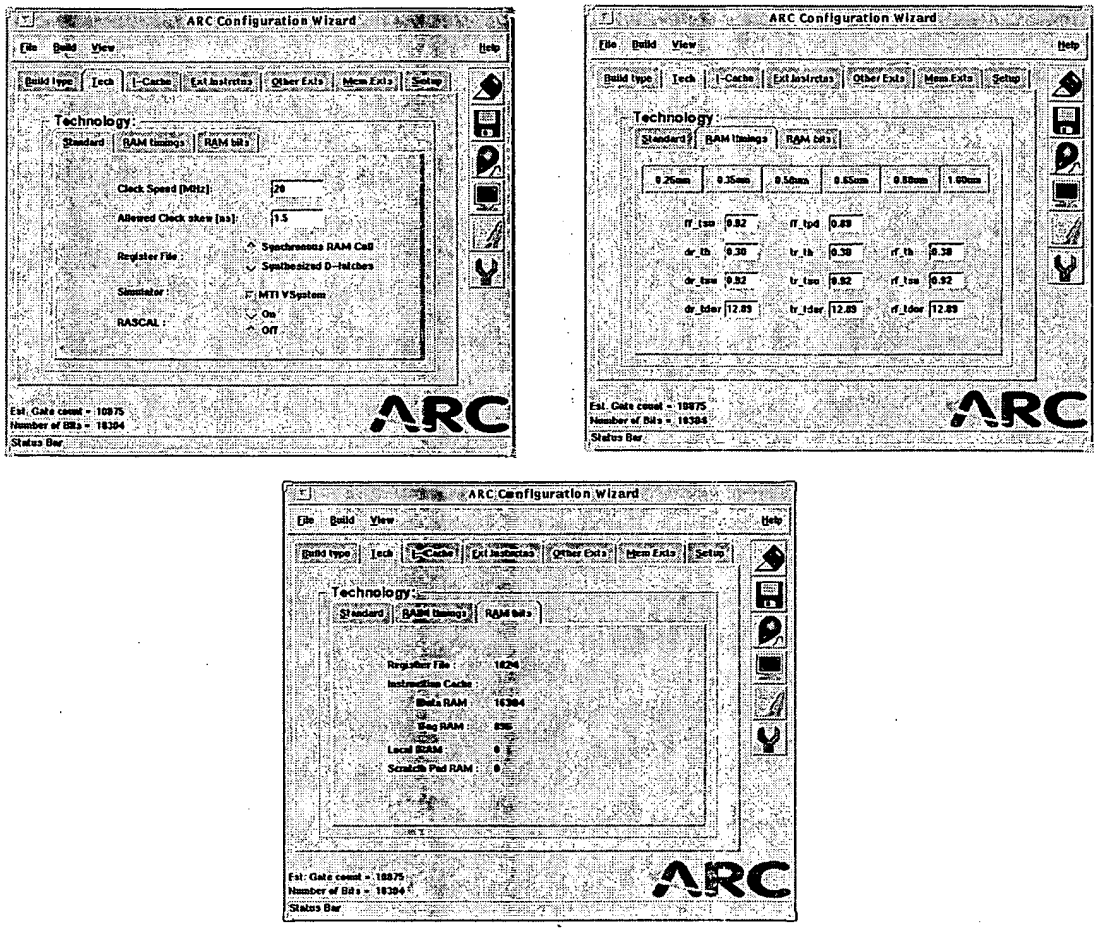


Figure 7: Technology Pages

On the technology page, a user has a choice of three sub menus, Standard, Timings and RAM bits.

Standard sub-page

On the standard sub-page, the user can set clock speed clock skew, type of register file and whether or not to create a ModelTech simulator makefile.

The user can enter values for Clock speed and Clock Skew in the appropriate entry boxes.

The register file section allows the user to select either synchronous 3 - port RAM (a fast write through is not required) or synthesised arrays of d-latches that require a write pulse to be generated. The standard synthesis script assumes a 90°-delayed clock signal.

Selecting the MTI simulator creates a makefile for use with MTI, and will automatically analyse all the VHDL as part of the build.

RTL And See-Code Application Link, otherwise known as RASCAL, is an extra utility that connects the Metaware debugger to the Modeltech simulator. This allows the debugger to control the simulation of code on the ARC system model in the MTI simulator for Hardware/Software Co-Verification. RASCAL can be enabled from this page.

Timings sub-page

On the timings sub-page, there is a button bar across the page. Each button in the bar represents a fabrication feature size. Pressing a particular button "fills in" several fields with default values. These fields include, timings for CELL, DATA / TAG ram, core registers and appropriate clock speed and skew. These are only guideline values based on a particular library for that feature size. The user can edit any of the settings by clicking in the appropriate entry box and entering a new value.

If after editing, for any reason, the user wishes to go back to the defaults, they can just click a feature size from the button bar.

WARNING, these values are passed unchecked to Synopsys during a build.

RAMBITS sub-page

This sub-page displays a breakdown of the Total RAMBITS displayed at the bottom of all pages. It shows the contribution to this total from the register file, Instruction cache, local RAM and Scratch Pad RAM.

NOTE:

- In the ARCAngel build, the register file is permanently set to synchronous single port Altera RAM. RAM timings are also disabled as ALTERA RAM timings are used.
- Clock speed and skew and timings are directly linked to the feature size of the technology you intend to use. Setting invalid timings can lead to invalid synthesis results.

3.6 Cache Page

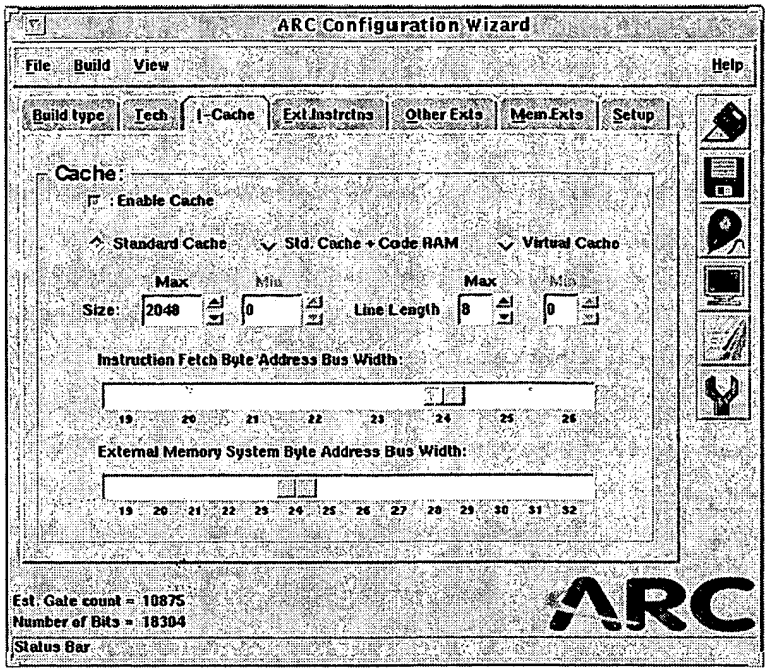


Figure 8: Cache Pages

At the top of the page is an "Enable Cache" button. When selected it allows the user to alter the cache settings. Otherwise, when the user builds the ARC, appropriate dummy files, for a 'no cache' build, will be included.

Standard Cache *with debug and instruction bypass modes,* is the "normal" version of the cache with the ability to bypass the cache altogether for debugging. Instruction Cache bypass allows the ARC to fetch required instructions from the data stream while on their way to the Instruction Cache, reducing the time penalty of loading a new line into the cache.

Virtual Cache allows the user to artificially reduce the Cache size and line length in software. This allows the user, using only one design, to run test code, and identify the most appropriate cache size for their final application. Having selected this mode, Max and Min values for cache size and line length have to be entered. The Virtual Cache is only intended for use in simulations and with the ARCAngel development board.

Standard Cache + Code RAM */ I-Cache space with Line Locking:* For more information, refer to the ARC interface manual.

Cache size. The Direct Mapped Instruction cache is user definable, by clicking on the up and down arrows a cache size can be selected. Sizes from 4 bytes (1 words), up to 512k bytes (130k words) are available. Other larger sizes can be manually instantiated after a build.

Cache Line length, is the number of words fetched when a cache miss occurs. The range of available cache line lengths is dependent on the cache size. Select a cache line length in the same manner as the cache size.

There are two (cache size and line length) boxes, Max and Min, if **Virtual Cache** is not enabled only the maximum (actual) size is definable. Refer to the Virtual Cache size section for more information.

Byte Address Bus Widths, Using the sliders you can select the appropriate byte address bus widths for both the Instruction fetch and the External memory.

NOTES:

- The Instruction Fetch Address Bus width CANNOT exceed the External Memory Address bus width and so the slider is restricted in this manner.
- If the Host communications port is set to 'PC/SUN parallel communications port' in the "Other extensions" page the sliders will not exceed 24 bits. This is because the maximum width of the PC port is 24-bits.
- In the ARCAngel build, both Address Bus widths are permanently set to 24-bits, as it is designed for use with the PC/SUN parallel communications port.

3.7 Extension Instructions

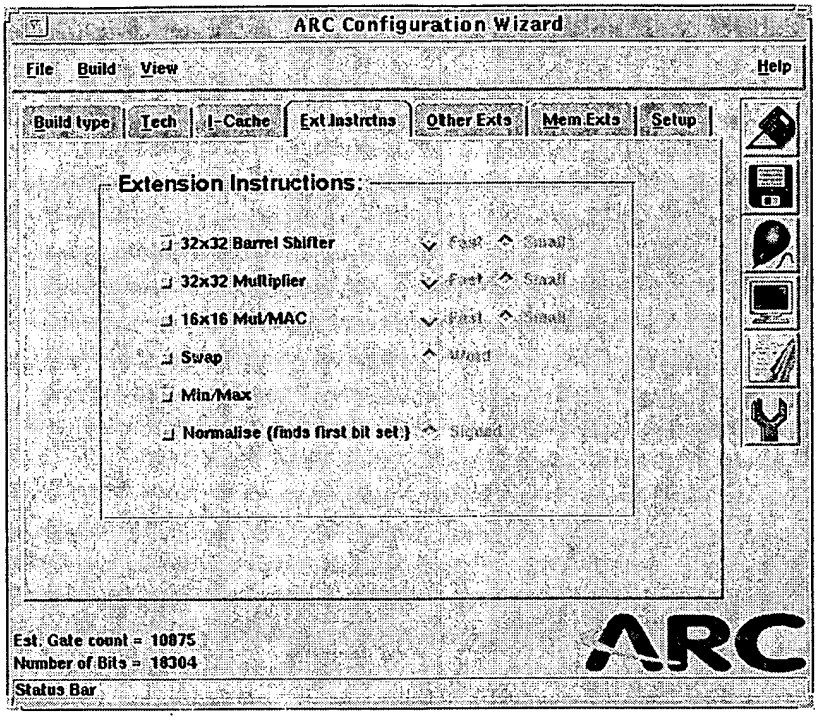


Figure 9: Extensions Page

On this page, the user can select combinations of the Extension instructions, by simply clicking on the corresponding buttons.

If the user selects the Barrel shifter, Multiplier or Mul/MAC they also have the option to select either a fast or a small version.

NOTES:

- Fast versions of the Multiplier and Mul/MAC are NOT available on the ARCAngel Build. This is due to the limited number of gates on the Altera chip, as such, you cannot select fast, and they default to small.

For more information about each individual extension, refer to the ARC Extensions Data sheets.

3.8 Other Extensions

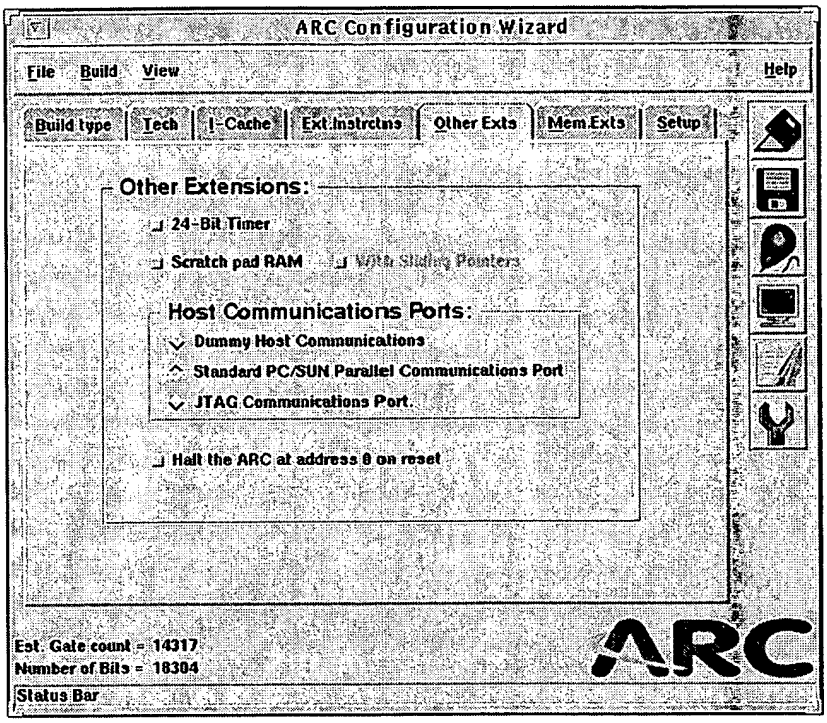


Figure 10: Other Extensions

This page provides more extensions, a 24-bit timer, and scratch pad RAM with or without sliding pointers. This page also contains host port selection, for generic builds, and the option to halt the ARC on reset.

The selectable Host communications port allows the user to select from **PC/SUN, JTAG or Dummy**. For the PC/SUN host communications port (default on an ARCAngel build), the `hostif.vhdl` file is set up in such a way that the ARC created will be able to communicate to a PC/SUN host via a parallel link. The Dummy host communications port, sets up `hostif.vhdl` as a template for the user to insert their VHDL to communicate to the host in their embedded application. While JTAG, (Joint Test Action Group) creates an ARC which communicates via the IEEE JTAG serial port. For more information on JTAG or PC port refer to the ARC interface manual.

Selecting the Halt button will halt the ARC at address 0 on reset. Different build types have different defaults for this option.

NOTES:

- The 24-Bit timer and Scratch pad ram, with and without sliding pointers, are not available on the core build.
- If, on the build page, the user selects *ARCAngel,* the 24-Bit timer is automatically selected. The 24-bit timer is only a suggested default for and can be deselected.
- The user can only choose the Host port during a *Generic* build. The host communications port has default of Dummy on the *Generic* build and standard PC/SUN on the *ARCAngel* build.
- If *Core* build has been selected, from the 'Build' page the wizard will automatically select 'Halt the ARC at address 0 on reset'. It can however be deselected.

3.9 Memory Extensions

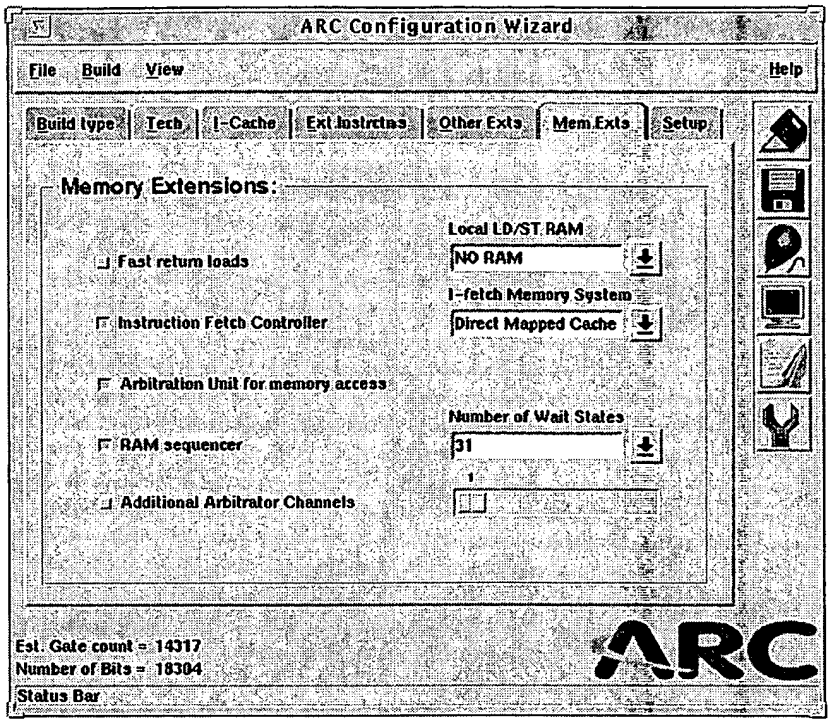


Figure 11: Memory Extensions

The Generic build gives you the ability to change settings on this page.

NOTE: Omitting any of these (excluding Additional Arbitrator Channels) will produce a design that will compile but will NOT simulate. Dummy blocks have been included to allow a successful VHDL compile, these dummy files are intended to be replaced by a users own VHDL designs.

Fast Load returns allows the use of shortcutting loads if the instruction does not use register write-back. Refer to the ARC programmers reference manual, or the release notes, for more information.

The **local LD/ST RAM** is a small area of memory, on chip, used for quick loads and stores. It has a variable base address specified by the contents of an auxiliary register.
The size of the 32-bit local RAM is user definable, with a range of zero to 16 K bytes. To change the size of the local RAM, click on the appropriate down arrow. A pull-down menu of all available local sizes will be displayed. Sizes from zero up to 16k bytes (4k words) are available. Clicking on a particular line in this pull-down menu will select the corresponding local ram size. For more info, refer to the ARC Extensions Data sheet.

The **Instruction Fetch Controller** fetches instructions from external memory and stores them in the instruction cache. If the user has chosen to build an ARC without and Instruction Cache the Instruction Fetch Controller is not needed and will be deselected and disabled. In addition, the Instruction Fetch Memory System will default to a dummy unit.

If the **Arbitration Unit** is not included, so that the user can include a custom design, both of the options to add additional Arbitrator channels and a SRAM sequencer channel are deselected and disabled.

The **SRAM Sequencer** is a state machine that reads/and writes to memory dependant on the stimuli from external memory interface and the memory arbitrator.

The number of SRAM sequencer wait states necessary for read/writes to memory is stored in an Auxiliary register. The default value for this register can be set by using the appropriate combobox.

If both the Memory Arbitrator and the SRAM Sequencer are omitted, a memory interface unit is included instead.

A user, when using the ARC Memory Arbitrator, may wish to add Additional Memory Arbitrator channels. This will allow any other designs, which access memory, to be used in conjunction with the ARC. When the Additional channels button is selected, the number of additional channels is determined by the moving slider bar. For each additional channel requested, the build puts dummy client VHDL files in the destination VHDL directory, "`client_chan0.vhdl`" etc.

For more detailed information on the memory extensions refer to the ARC interface manual.

3.10 Set-up

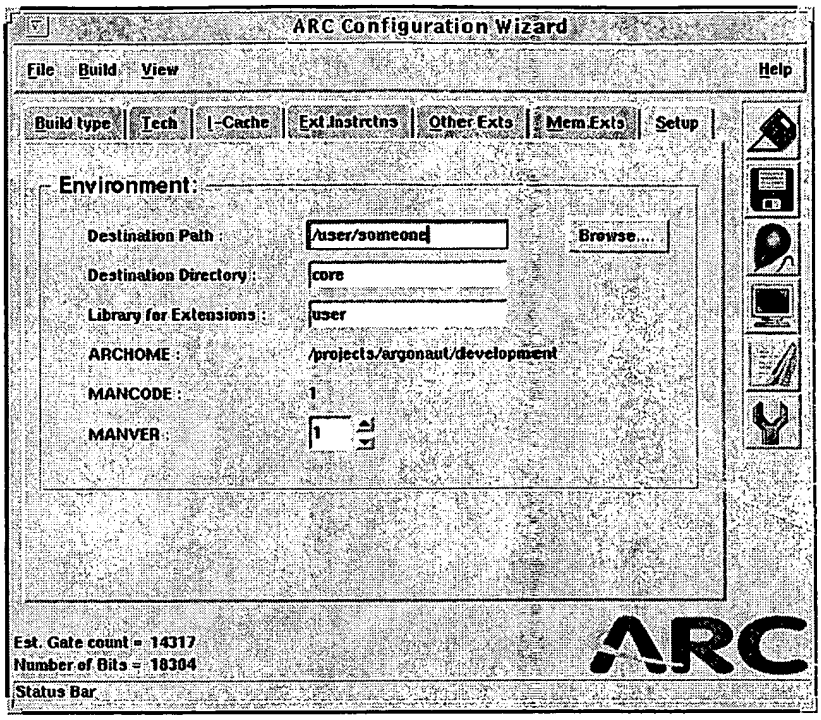


Figure 12: Set-up Page

Before building the ARC a destination for the files must be specified. On the setup page the user can specify the Destination Directory, the name of the directory that build files will be placed in and the Destination Path, which is the directory below the Destination Directory.

It is possible to specify the Destination Path by either typing directly in the appropriate entry box, or by clicking on the Browse button, that will create a directory tree dialogue box. The tree can then be traversed in the usual manner. When the correct Destination Path is displayed in the top entry box, clicking on 'OK' returns.

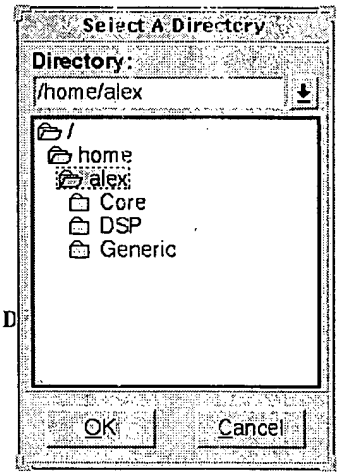


Figure 13: Directory D

The 'Library for Extensions' entry box allows the personalisation of user library directories. The string entered in this box is used during the install when creating two user library directories and copying subsequent files to these directories, (The directories created use the string in upper case).

Also displayed on this page are the environment variables, used during the install. If any of these say "NOT SET' the user should save their settings, quit the wizard and then restart in a shell that has these environment variables set.

3.11 Summary

Before starting an ARC build, the configuration can be checked quickly and easily from the menu bar. Located along the top of the wizard, the user can select `View -> View Summary`, or press `'CTRL +V'` for a shortcut. This displays a convenient summary of all selections.

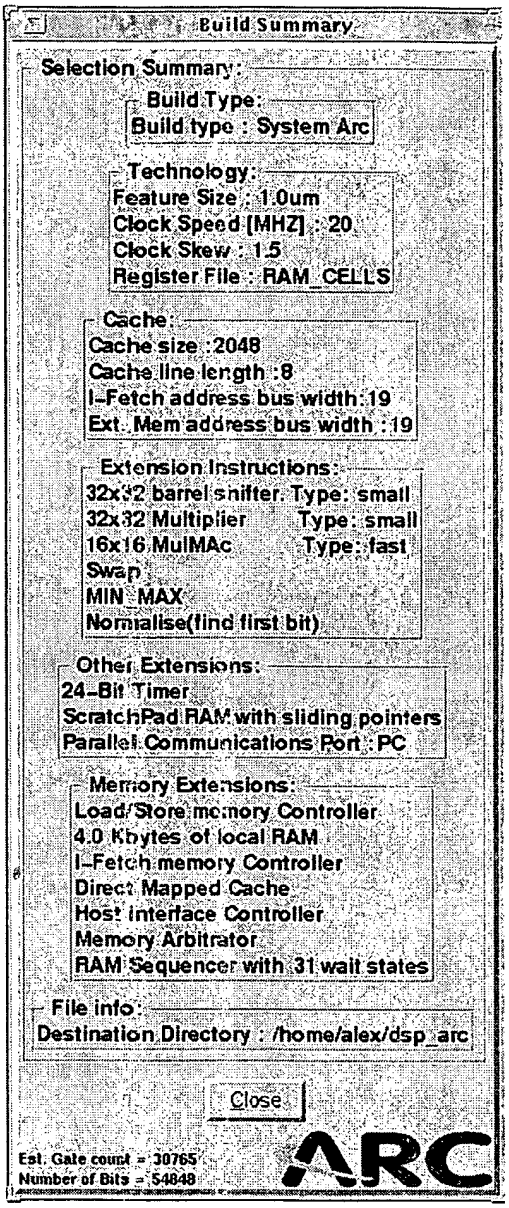
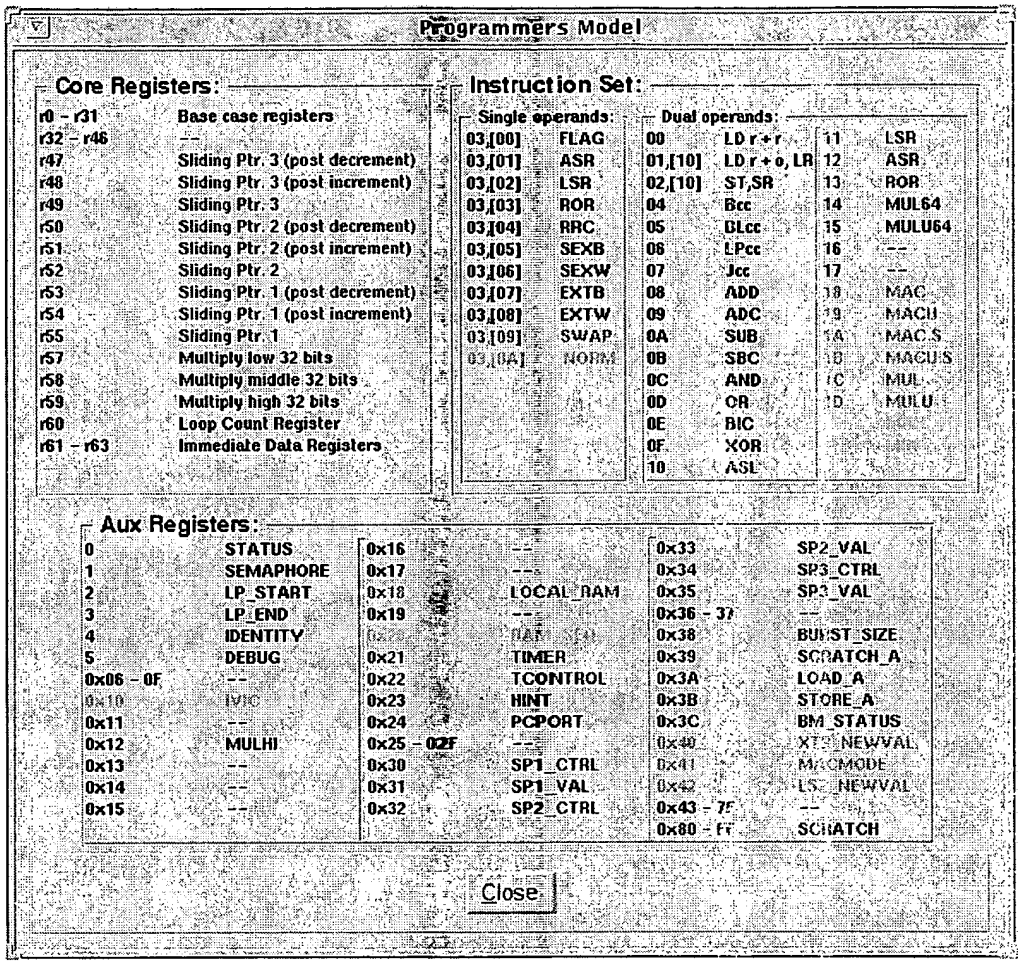


Figure 14: Summary Dialogue box

3.12 Programmers Model



The ARC programmers model is a display of the core registers, auxiliary registers, and instruction set. The ARC programmers model is dynamic, as extensions are added the associated registers and opcodes are added. Each extension is colour coded, e.g. The 16x16 Mul/Mac is CYAN, has six op-code entries and three associated auxiliary registers.

System Builder User Guide

Document revision - v 1.9

Revision Information

Author:	Alexander Holland
Last Revised By:	Mohammed Khan
Revision:	1.9
Date of last revision:	31-July-98
Filename:	sbug.pdf

Contents

List of Figures

Preface

This document describes the ARC System builder, the build script used for creating files for simulation on a VHDL simulator and VHDL synthesis.

This document is aimed at ASIC engineers working with the ARC.

1 - What is the ARC System Builder?

1.1 Introduction

The ARC System builder is a script that allows users to build customised ARC systems along with support files for simulation and synthesis. When executed the build script produces a series of questions, mainly answered through menus, the answers to which are used to build the VHDL simulator and synthesis files.

- Installation script allowing the following ARC features to be selected:
  - ◊ Extensions required
  - ◊ Cache size
  - ◊ Cache line length
  - ◊ Size of external memory space that is to be cached.
  - ◊ Clock period
  - ◊ Clock skew
  - ◊ Synthesised d-latches or 3-port RAM for register file
  - ◊ Type of technology being used.
  - ◊ Manufacturer code and version number information

- The script creates a working directory for the user, and in it creates the following:
  - ◊ VHDL for the extensions selected
  - ◊ VHDL for a direct mapped I-cache configured to user's specification
  - ◊ VHDL test-bench for testing basecase operation and external interfaces
  - ◊ VHDL structure to link together all required modules
  - ◊ Configuration files for Synopsys Design Compiler
  - ◊ Memory image file containing basecase test code
  - ◊ Synthesis script for Synopsys Design Compiler v3.4b or above
  - ◊ Makefile - set up for the Model Technologies VSystem/VHDL simulator, but can be altered for use with other simulation environments.

- The user can select from 4 different types of build:
  - ◊ Core Build
  - ◊ Generic System Build
  - ◊ Altera Build (for ARCAngel development board)
  - ◊ Core Verification System

2 - Using the ARC System builder.

2.1 Setup

After setting up your ARC area, before you can use the *ARC system builder*, several environment variables need to be set.

In .cshrc:
Set **ARCHOME** environment variable, base of the arc install tree. Add bin directories to path. Set other environment variables such as **ARC_MANCODE** and **ARC_MANVER** and **ARCASIC**. (If these variables are not set the build script will ask for them at run time.)

e.g.

```
setenv ARCHOME /apps/argonaut
setenv ARC_MANCODE <Users Manufacturer code>
setenv ARC_MANVER <Users selected version number>

set path = ($ARCHOME/arc/bin $path)
```

In order to use the System Builder with ModelTech VSystem, Synopsys Design compiler or Max Plus II (required when building for the ARCAngel development board), the Shell in which you intend to run the System builder must be set-up to use the specific tool.

ARC_MANCODE is the unique manufacturer code that Argonaut gives to each ARC licensee, and the **ARC_MANVER** is that manufacturers additional identity number. Both numbers go on to be stored as part of the identity register. (For more information on the identity register see the ARC Programmers Reference Manual.)

2.2 Launching the ARC system builder.

After setting up your environment, the ARC system builder can be run just by typing `system_builder`, which is an executable script.

A log file (`sys_bld.log`), which keeps a record of the build selections you make, is created in the directory that you specify for the user working directory.

See appendix for command line information.

2.3 Menus

The ARC System builder script when run produces a series of text menus/questions. (See Appendix **Menu Structure** for more details.)

```
Example MENU 1

   1. Continue.
   2.   Valid selection.
   3. * Mutually exclusive.
   4.   Mutually inclusive.
   -  * Permanently selected.
 N/A    Invalid selection.
   7. * Valid selection
Please select :
```

Figure 1 Example of menu type 1.

The above menu shows an example menu. The '*' next to a menu entry indicates that it is selected.

Continue

On all the menus that allow multiple selections **1** is always continue.

Valid selections

In this example menu entry **2** is a valid selection which is currently unselected. Menu entry **7** is also a valid selection, which is selected. Valid selections can be selected and deselected by entering their corresponding menu number. (When menus are initially displayed some valid selections may be pre-selected; these are just recommended selections for this build.)

Mutually exclusive selections (On the same menu)

Selections are mutually exclusive, for example two *types* of the same extension cannot be selected at the same time. If one menu entry, of a mutually exclusive set, is already selected and the user then selects the another, the first is deselected.

Mutually inclusive selections (Between menus)

Mutually inclusive relationships can exists between menu entries on two separate menus. If this is so, the entry on the second menu is either permanently selected or invalid depending on the state of the first entry.

Permanent selections

In certain circumstances menu entries usually valid select/deselect are permanently selected, e.g. Example menu entry **5**. This usually because on a previous menu the user selected a menu entry to which this one is mutually inclusive. Permanent selections can also be active due to the build type.

Invalid selections

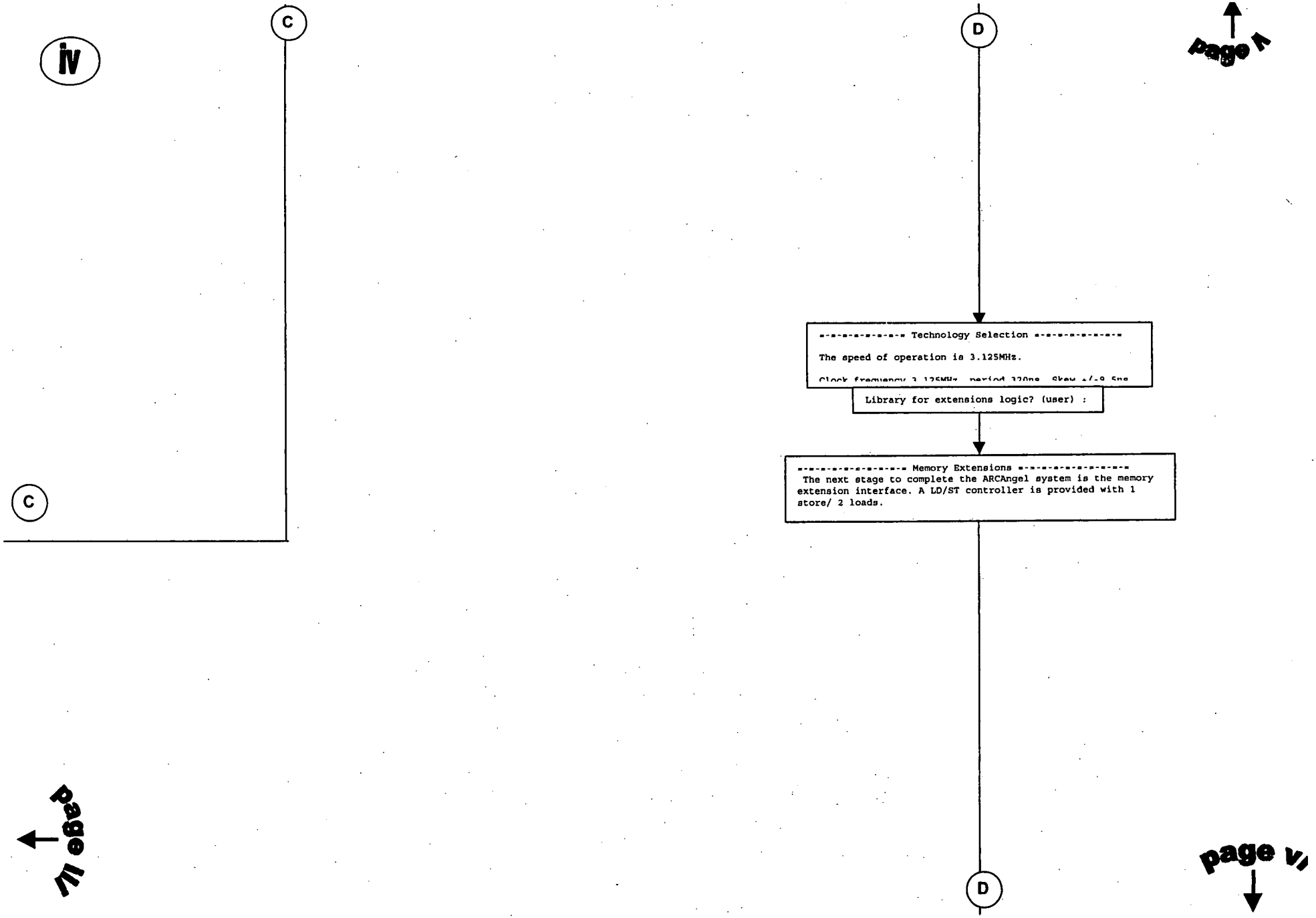
As with permanent selections, invalid selections are normal valid select/deselect menu entries, however in this case these options are not available. Again this is usually due to either mutually inclusive selections on previous menus that were not selected, or this selection is not available on this type of build. Menu entry **6** is an example of an invalid selection.

```
Example MENU 2

   1.   Selection 1.
   2.   Selection 2.
   3.   Selection 3.
   4.   Selection 4.

Please select(3) :
```

Figure 2 Example of menu type 2

Menu Default

Some menus in the *ARC system builder* have default selections. On such menus entering nothing, i.e. just pressing enter, will select the default. The default menu entry number is specified next to the requestor. Menu defaults can be more than just numbers, on some questions they can be file paths, y/n etc.


iv
C
C
D
page v
=-=-=-=-=-=-=-= Technology Selection =-=-=-=-=-=-=-=
The speed of operation is 3.125MHz.
Library for extensions logic? (user) :
=-=-=-=-=-=-=-=-= Memory Extensions =-=-=-=-=-=-=-=-=
The next stage to complete the ARCAngel system is the memory extension interface. A LD/ST controller is provided with 1 store/ 2 loads.
D
page vii
page vi

3 - Walkthroughs

This section contains a quick walkthrough for each of the four types of build available, Base case, Generic, ARCAngel and Core verification.

The first menu you will encounter for all installations is the one below.

```
ARC System Build Tool - ARC System install.
(c) 1998 Argonaut Technologies Ltd.

Generic install
 This program is capable of creating a basecase ARC or a complete ASIC for
simulation, using MTI and synthesis using Synopsys Design Compiler/Design
Expert.

We're going to make :

   1.  The directory structure
   2.  The structural VHDL levels
   3.  A personalised version of the cache config package
   4.  An optional external memory interface
   5.  Synopsys RAM models for the cache RAMs
   6.  Synthesis scripts
   7.  A simulation makefile

But first, you must answer some questions :

Please select the type of system you wish to build.

   1.  Core build with selected extensions
   2.  Generic System Build with selected core/memory extensions
   3.  Altera Build for ARCAngel Development Board
   4.  Core Verification System.

Please select (1) :
```

It allows the user to select from four types of build Core (Base case) build, Generic, ARCAngel and Core Verification. We will now walkthrough an install of all four types, showing the menus you will encounter and 'typical' responses.

3.1 Base case

From the first menu we select **1**, which is also the default. The ARC system builder will then prompt the user for details about the Core build, i.e. Standard extensions (not including any scratchpad extensions), Direct mapped instruction cache, Load/ST interface to external memory, Host interface and model of external memory.

```
=-=-=-=-=-=-=-=-=-= Instruction/Data byte Address selections =-=-=-=-=-=-=-=-=-=-=-

Instruction Fetch memory system byte address bus width

   1.  22 bit (4Mb)
   2.  23 bit (8Mb)
   3.  24 bit (16Mb)
   4.  25 bit (32Mb)
   5.  26 bit (64Mb)

Please select (3) :
```

- This menu asks for the number of bits for the instruction fetch address. For the purpose of this walkthrough we will select the default of 24 bit (16Mb), by pressing ENTER.

```
The external memory system data bus width is 32 bits wide
```

The external data bus on the ARC is 32-bits wide.

```
External memory system address bus width (22 < address < 32)(24) :
```

- The external memory system data bus width can range from 22 to 32 bits, 4Mb to 4Gb address range, the default on this menu is set to the value of the I-fetch address bus width. We are going to select **29** bits, an address range of 512 Mb.

NOTE: Even though the bus width is set to an arbitrary value the memory model for a core build simulation is 128Kbytes.

```
=-=-=-=-=-=-=-=-=-=-=-=-= Core Register File Selection =-=-=-=-=-=-=-=-=-=-=-=-=-

 The 32x32 register file can be implemented using a synthesised array
of d-latches, or by using a three-port (2 read, 1 write) synchronous
RAM cell.

Use a synchronous 2r1w RAM for the register file? (y) :
```

- For the 32 general purpose registers r0-r31 you can either use :-
  (y) Synchronous 3-port (2r1w) RAM, (a fast write-through is not required).
  (n) Synthesised arrays of d-latches that require a write pulse to be generated.
- Standard synthesis script assumes a 90°-delayed clock signal, for the synthesised array of d-latches.

We've got a synchronous 3-port RAM and so we select **y**.

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= Fast Load Returns -=-=-=-=-=-=-=-=-=-=-=-=-=-=-

 Enable Fast load returns? (n) :
```

Fast Load returns, if the current instruction does not use register write back then the pipeline is not stalled.

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= Extensions Builder =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

Select the extensions to be added to the ARC

   1.  Continue
   2.    32x32 Barrel shifter - rotate, arithmetic and logical shifts - 1 cycle
   3.    32x32 Barrel shifter - rotate, arithmetic and logical shifts - Multi-cycle
   4.    Small multi-cycle 32x32 Multiply
   5.    Faster (& larger) 32x32 Multiply
   6.    16x16 Multiply/Multiply Accumulate function with 36 bit accumulator - Fast
   7.    16x16 Multiply/Multiply Accumulate function with 36 bit accumulator - Slow
   8.    Swap function - swaps upper and lower 16 bits of 32 bit word
   9.    MIN/MAX function - 32 bit precision
   10.   Normalise function - finds first bit set in 32 bit word
   11.   24-bit Timer which counts to limit and then generates an interrupt
  N/A    A Scratchpad RAM
  N/A    A Scratchpad RAM with Sliding Pointers

Please select :
```

- The above menu shows all the standard extensions, you can select lots of different combinations depending on your needs.
- We will select the single cycle barrel shifter and the 32x32 multiply. Entering **2** and then entering **4** does this. If we wanted to disable the multiply we could do this by entering **4** again.
- Note you cannot select both types of barrel shifters, in an install, selecting one automatically deselects the other. This also applies to both instances of the 32x32 multiplier, the MAC and both instances of the Scratchpad RAM.
- We will now enter **1** to continue.

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= Technology Selection =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

Feature size (drawn) of technology in use

   1.  0.35um
   2.  0.5um
   3.  0.65um
   4.  0.8um
   5.  1.0um

Please select (5) :
```

- The feature size provides timings for DATA / TAG ram and the core registers. Our chosen technology is 1.0µm so press ENTER to continue.
- The default synthesis script is set-up for the 1.0µm LSI_10K library supplied with the Synopsys synthesis tool.
- These values are only used to set the timings for the synthesis of RAM models. If an unsupported feature size required you can select an arbitrary value and following the build change the clock period for the simulation of the VHDL and synthesis scripts to support the new feature size, timings of RAM etc.

```
Target clock speed in MHz (20)
```

- We will use the default of **20MHz** as we have specified 1.0 µm technology.

Clock speed has a minimum of 1 MHz and a Maximum of 150 MHz.

```
Clock skew for synthesis (+/-1.5ns) :
```

- The default of **1.5ns** will be used for this walkthrough.

Clock Skew has a minimum of 0 and a maximum less than the clock period.

```
Library for extensions logic? (user) :
```

- Here you can specify a name for your user's library. This name will be used as the VHDL library names and also during the install when creating directories and copying subsequent files to these directories (The directories created are in upper case). The name entered is arbitrary so we will enter 'alex', why not?

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= Memory Extensions =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

 Do you wish to include a Direct-Mapped Instruction Cache? (y) :
```

- If you enter "no" here you will build an ARC with no cache, if you enter "Yes" you will be prompted to for further information about the cache you want. We will enter "Yes" the default.

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= Ifetch Modes =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

 Select the instruction Cache Modes.

   1. Continue
   2. Standard Cache with debug and cache bypass capability.
   3. Virtual Cache
   4. Mixed Code RAM / Icache space & Cache line lock
```

- The first thing the script needs to know is what kind of cache do you want. It's possible to have a :
    1. Standard Cache
    2. Virtual cache allows a user to dynamically resize the cache in software.
    3. Mixed Code RAM / Icache space with Line locking

For more information on Instruction cache modes refer to the ARC interface manual.
NOTE: Standard Cache and Virtual Cache are mutually exclusive, and Mixed Code RAM is a superset of Standard Cache. We want to try a range of cache sizes to see what cache size best suits our application, therefore we are selecting Virtual Cache

```
 Please select the max size of the Direct Mapped Instruction Cache.

   1. 0.5k bytes, 128 words
   2. 1k bytes, 256 words
   3. 2k bytes, 512 words
   4. 4k bytes, 1024 words
   5. 8k bytes, 2048 words
   6. 16k bytes, 4096 words
   7. Other size

Please select (3) :
```

- The Direct Mapped Instruction cache is user definable, using the script you can select sizes from 4 bytes (1 word), up to 512k bytes (128k words). There are shortcuts for 512 bytes to 16K bytes, but other sizes (larger or smaller) can be entered manually using option 7. For our test 4K bytes (selection 4) is sufficient.

```
Please select the min size of the Direct Mapped Instruction Cache.

   1. 0.5k bytes, 128 words
   2. 1k bytes, 256 words
   3. 2k bytes, 512 words
   4. 4k bytes, 1024 words
   -  8k bytes, 2048 words
   -  16k bytes, 4096 words
   7. Other size

Please select (3) :
```

- Virtual Cache requires a range of possible caches and line lengths so a minimum cache size is required. We will enter 2 512 bytes.

```
Select a max Cache line length

  1.  2 instruction words
  2.  4 instruction words
  3.  8 instruction words
  4.  16 instruction words
  5.  32 instruction words
  6.  Other

Please select (3) :
```

- The cache line length is the number of words fetched when a cache miss occurs. The script has options for 2, 4, 8, 16, or 32 word lines, the default being **8 words.** Selecting **4**, will give a cache line length of 16 words, suitable for our test.

```
Select a max Cache line length

  1.  2 instruction words
  2.  4 instruction words
  3.  8 instruction words
  4.  16 instruction words
  -   32 instruction words
  6.  Other

Please select (3) :
```

- Virtual Cache requires a range of possible caches and line lengths so a minimum cache line length is required. Entering **1**, selects two instruction words.

Cache size, line length and external address all determine the size of the tag RAM. Larger cache → more tag words fewer tag bits. Longer line length → fewer tag words more tag bits.

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= Simulation =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

Selection of the state of the ARC on reset is important since it is preferable
to have the system build start on reset while the core build should halt on
reset since the host effectively clears the halt bit during simulation.

 Do you wish to halt the ARC at address location 0 on reset (y) :
```

- We have entered '**y**' here, that means the ARC is halted immediately on reset. Had we selected '**n**' the ARC starts running the code at address 0 on reset.
- The default for this is Y on core build and N on generic and ARCAngel builds.

```
Note you will have to set the status register to the appropriate start address
and clear the halt bit through the host interface before running the simulation.
You can manually edit "arc_start" in  xaux_regs.vhdl so that the ARC is running
following a reset.
```

- Having selected '**y**' the above information message is displayed.

```
Is your VHDL simulator Model Technologies VSystem? (y) :
```

- At Argonaut Technologies we enter '**y**' as we use MTI VSystem to simulate our VHDL. If you use a different simulator, select no. You may set your environment up for your simulator by editing the makefile and other support files, after the install has completed.

```
Do you wish to use the R.T.L. And SeeCode Application Link (RASCAL)? (n) :
```

- **R**TL **A**nd **S**ee-**C**ode Application **L**ink, otherwise known as RASCAL, is extra utility that connects the Metaware debugger to connect to the Modeltech simulator. This allows the debugger to control the simulation of code on the ARC system model in the MTI simulator for Hardware/Software Co-Verification. The default is no which is what we want for this build.

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= Install Directory =-=-=-=-=-=-=-=-=-=-=-=-=-=-=

Full path to ARC user working directory (/home/someone/basearc) :
```

- At this prompt you have to specify a destination directory name (with full path).
- The path must not include /tmp_mnt/ etc.
- If the directory currently exists, it will not be overwritten, you will just be prompted for another path.
- We will select the default, a directory named "`basearc`" created where we launched the ARC system builder.

What we have built

I-fetch address bus width	24-bit	
external address bus width	29-bit	
register file	3-port RAM	
Fast Load returns	'no'	
Extensions	Single Cycle Barrel shifter	
	Small multi-cycle 32x32 multiply	
Clock speed	20 MHz	
Clock skew	1.5 ns	
Technology size	1.0 µm	
Users library name	'alex'	
Direct mapped instruction	Virtual	
Max/ Min instruction cache size	4K bytes	512 bytes
Max/ Min Cache line length	16 instruction words	2 instruction words
Halt ARC on Reset?	'yes'	
Destination directory	`/home/user/basearc`	
Simulator = MTI?	`'yes'`	
RASCAL	`'no'`	

3.2 Generic

The second selection from the first menu is the generic build. This is similar to the base case build however it allows the selection of memory extensions. As said earlier in the generic build certain selections determine which menus you will see, we are going to make selections in this walkthrough that will give access to all menus.

```
=-=-=-=-=-=-=-=-=-= Instruction/Data byte Address selections =-=-=-=-=-=-=-=-=-=-=-

Instruction Fetch memory system byte address bus width

   1.  22 bit (4Mb)
   2.  23 bit (8Mb)
   3.  24 bit (16Mb)
   4.  25 bit (32Mb)
   5.  26 bit (64Mb)

NOTE: If you intend to use the PC parallel communications host port do not exceed 24-bits

Please select (3) :
```

- This menu asks for the number of bits for the instruction fetch address. For the purpose of this walkthrough we will select 22 bit (4Mb), by entering **1**.

NOTE: Later in the build you will be asked to select a host port for communications with the ARC. The SUN/PC parallel communications host port has maximum of 24-bits, if you select a value greater than 24-bits in this menu the PC host port will no longer be available.

```
The external memory system data bus width is 32 bits wide
```

The external data bus on the ARC is set to 32 bits wide.

```
External memory system byte address bus width (22 < address < 32)(22) :
```

- The external memory system data bus width can range from 22 to 32 bits, 4Mb to 4Gb address range, the default on this menu is set to the value of the I-fetch address bus width. We are going to select **24** bits, 16Mb address range, which is the maximum valid selection that can be used with the PC/SUN host port.

NOTE: Even though the bus width is set to an arbitrary value the memory model for a generic build simulation is 512Kbytes.

```
=-=-=-=-=-=-=-=-=-=-=-=-= Core Register File Selection =-=-=-=-=-=-=-=-=-=-=-=-=-

 The 32x32 register file can be implemented using a synthesised array
of d-latches, or by using a three-port (2 read, 1 write) synchronous
RAM cell.

Use a synchronous 2r1w RAM for the register file? (y) :
```

- For the 32 general purpose registers r0-r31 you can either use :-
  (y) Synchronous 3-port (2r1w) RAM, where a fast write-through not required, or
  (n) Synthesised arrays of d-latches that require a write pulse to be generated.
- Standard synthesis script assumes a 90°-delayed clock signal.

We've got a synchronous 3-port RAM and so we select **y**.

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= Fast Load Returns -=-=-=-=-=-=-=-=-=-=-=-=-=-

 Enable Fast load returns? (n) :
```

Fast Load returns, if the current instruction does not use register write back then the pipeline is not stalled. For more information on Fast load returns refer to the release notes. Our design allows us to take advantage of fast return loads and therefore we enter 'yes'.

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= Extensions Builder =-=-=-=-=-=-=-=-=-=-=-=-=-=-=

Select the extensions to be added to the ARC

   1.  Continue
   2.    32x32 Barrel shifter - rotate, arithmetic and logical shifts - 1 cycle
   3.    32x32 Barrel shifter - rotate, arithmetic and logical shifts - Multi-cycle
   4.    Small multi-cycle 32x32 Multiply
   5.    Faster (& larger) 32x32 Multiply
   6.    16x16 Multiply/Multiply Accumulate function with 36 bit accumulator - Fast
   7.    16x16 Multiply/Multiply Accumulate function with 36 bit accumulator - Slow
   8.    Swap function - swaps upper and lower 16 bits of 32 bit word
   9.    MIN/MAX function - 32 bit precision
   10.   Normalise function - finds first bit set in 32 bit word
   11.   24-bit Timer which counts to limit and then generates an interrupt
   12.   A Scratchpad RAM
   13.   A Scratchpad RAM with Sliding Pointers

Please select :
```

- The above menu shows all the standard extensions, you can select lots of different combinations depending on your needs.
- We will select multi-cycle barrel shifter, Fast Mul/MAC, scratchpad RAM with sliding pointers and swap. Entering **3,7,12** and **8** does this. We really wanted a Normalise extension rather than Swap. We disable Swap by entering **8** again and then enter **10** to select normalise.
- Note that you cannot select both types of barrel shifters, in an install, selecting one automatically deselects the other. This also applies to both instances of the 32x32 multiplier, the MAC and both instances of the Scratchpad RAM.
- We will now enter **1** to continue.

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= Technology Selection =-=-=-=-=-=-=-=-=-=-=-=-=-=-=

Feature size (drawn) of technology in use
   1.  0.35um
   2.  0.5um
   3.  0.65um
   4.  0.8um
   5.  1.0um

Please select (5) :
```

- The feature size provides timings for DATA / TAG ram simulation & synthesis models and the core registers. Our chosen technology is 0.35µm so press **1** to continue. If vendor models are to be used the timings these settings produce are not used.
- These values are only used to set the timings for the synthesis of RAM models. If an unsupported feature size required you can select an arbitrary value and following the build change the clock period for the simulation of the VHDL and synthesis scripts to support the new feature size, timings of RAM etc.

```
Target clock speed in MHz (100)
```

- We will use the default of **100Mhz** as we have specified 0.35 µm technology.

```
Clock skew for synthesis (+/-0.4ns) :
```

- The default of **0.4ns** will be used for this walkthrough.

```
Library for extensions logic? (user) :
```

- Here you can specify a name for your user's library. This name will be used as the VHDL library names and during the install when creating directories and copying subsequent files to these directories. The name entered is arbitrary so we will enter 'sam'.

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-= Memory Extensions =-=-=-=-=-=-=-=-=-=-=-=-=-=

Select the memory extensions to be added to the ARC interface

   1.  Continue
   -   * Load/Store memory controller
   3.  * Instruction fetch memory controller
   4.  * Host interface for communications with ARC
   5.  * Arbitration unit for memory accesses
   6.  * RAM Sequencer

Please select :
```

- Here the memory extensions for off chip memory are selected. The LD/ST memory controller is not optional, and is therefore permanently selected. To show the full use of the build script we are selecting all memory extensions. To do this there must a '*' next to each number, if not type that number (and pressing enter) , finally enter 1 to continue.

```
The queue depth for the Load/Store Memory Controller is fixed, i.e. 4 deep
with a max of 2 stores.
```

- Having continued the above information message is displayed.

```
-================= Ld/St Memory Controller ==================-

Select RAM for local load/store operations.

   1.  No Local RAM
   2.  512  x 32 bit of Local RAM
   3.  1024 x 32 bit of Local RAM
   4.  2048 x 32 bit of Local RAM
   5.  4096 x 32 bit of Local RAM

Please select (3) :
```

- The size of RAM to be used for local memory can be specified. We want 8 Kbytes of 32 bit local ram and so enter 4. The local RAM is situated at the top of memory by default (determined by the external address width) and therefore has a base address of [Maximum RAM size - Local RAM size].

```
-================ I-Fetch Memory Controller ================-

Select type of Instruction Fetch memory system

   1.  Dummy I-fetch unit for User Definition
   2.  Direct Mapped Cache

Please select (2) :
```

- There are several different ways in which the instruction fetch memory system can be defined.

1. Dummy I-fetch, creates a template VHDL file for the user to edit.
2. Direct mapped cache.

We are using the default, 2, Direct Mapped Cache. (For more information on Instruction Fetch Cache refer to the ARC Interface manual)

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= Ifetch Modes =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

 Select the instruction Cache Modes.

   1. Continue
   2. Standard Cache with debug and cache bypass capability.
   3. Virtual Cache
   4. Mixed Code RAM / Icache space & Cache line lock

Please select :
```

- The first thing the script needs to know is what kind of cache do you want. It's possible to have a :
    1. Standard Cache
    2. Virtual cache allows a user to dynamically resize the cache in software.
    3. Mixed Code RAM / Icache space with Line locking

We have previously looked at using Virtual Cache to experiment with a range of cache configurations. Using this method we have found a standard cache configuration that gives the required performance to cost ratio for our application. We are therefore selecting Standard Cache.

For more information on Instruction cache modes refer to the ARC interface manual.
NOTE: Standard Cache and Virtual Cache are mutually exclusive, and Mixed Code RAM is a superset of Standard Cache.

```
 Please select the max size of the Direct Mapped Instruction Cache.

   1. 0.5k bytes, 128 words
   2. 1k bytes, 256 words
   3. 2k bytes, 512 words
   4. 4k bytes, 1024 words
   5. 8k bytes, 2048 words
   6. 16k bytes, 4096 words
   7. Other size

Please select (6) :
```

- The Direct Mapped Instruction cache is user definable, using the script you can input select sized from 512 bytes (128 words), up to 16k bytes (4k words). Other sizes (larger or smaller) can be manually input using option 7. For our test 8K bytes (selection 5) is sufficient.

```
Please select the min size of the Direct Mapped Instruction Cache.

   1. 0.5k bytes, 128 words
   2. 1k bytes, 256 words
   3. 2k bytes, 512 words
   4. 4k bytes, 1024 words
   5. 8k bytes, 2048 words
   -- 16k bytes, 4096 words
   7. Other size

Please select (1) :
```

- As we are using a virtual cache, it is necessary to specify a lower limit for the cache size. When using a virtual cache it is possible to artificially set the cache to any size between the maximum and minimum using software. For this test we want to test a large range of cache sizes down to the very small, therefore we select 7.

```
Please enter the min size of the Direct Mapped Instruction Cache in bytes.
NOTE: Size must be a power of two also 4 <= Size <= 16384

Enter Size :
```

- As we are testing a large range of caches we have decided to choose 128 bytes as our minimum cache size. Entering **128** and pressing enter sets this constant. At this menu we can use abbreviations, 16k for example is the same as entering 16384.

```
Select a max Cache line length

  1.  2 instruction words
  2.  4 instruction words
  3.  8 instruction words
  4.  16 instruction words
  5.  32 instruction words
  6.  Other size

Please select (3) :
```

- The cache line length is the number of words fetched when a cache miss occurs. The script has options for 2, 4, 8, 16, or 32 word lines, the default being **8 words.** Typing **ENTER** here, like all other menus, selects the default, **3**, giving a cache line length of 8 words.

```
Select a min Cache line length

  1.  2 instruction words
  2.  4 instruction words
  3.  8 instruction words
  --  16 instruction words
  --  32 instruction words
  6.  Other size

Please select (3) :
```

- When using a virtual cache, as with the cache size, you must also specify a minimum cache line length. **1** will give a minimum line length of 2 instruction words, which is the choice we have made.

Cache size, line length and external address, all go to determine the size of the tag RAM. Larger cache → more tag words fewer tag bits. Longer line length → fewer tag words more tag bits.

```
-============= Host Interface Communications ================-

Select from available host communications ports.

   1.  Dummy Host Communications Port
   2.  Standard PC/SUN Parallel Communications Port
   3.  JTAG Communications Port

Please select (2) :
```

- Host port for I/O communications can be either a dummy VHDL file, which you can edit to create a user defined communications port, standard PC/SUN or JTAG communications ports. We are selecting **2**, PC/SUN, as we are not using our ARC with custom hardware.

```
-======================== Memory Arbitrator ========================-

Select the channels that you would like the memory arbitrator to service.


   1.  Continue
   -    * Load/Store channel for external memory accesses.
   -    * Host Interface channel for debugging and data downloading.
   -    * Instruction fetch channel for I-cache.
   -    * Local Scratch-pad RAM channel for burst reads/writes.
   6.     Additional Channels

Please select :
```

- How this menu looks is very dependent on the selections that have been made before, as most options are mutually inclusive with other selections. For test purposes we are adding some additional channels **6**.

```
How many additional channels do you want? (1 - 10) :
```

- The user can select from between 1 and 10 extra channels. Enter **5** to give 5 additional channels.

```
1.  Continue
   -   * Load/Store channel for external memory accesses.
   -   * Host Interface channel for debugging and data downloading.
   -   * Instruction fetch channel for I-cache.
   -   * Local Scratch-pad RAM channel for burst reads/writes.
   6.  * Additional Channels(5)

Please select :
```

- The number of additional channels is now displayed next to the option. We enter **1** to continue to the next menu.

```
-====================== RAM Sequencer ========================-

Select the number of wait states for the RAM Sequencer. The
script does not allow the user to enter a value greater than
31. If you wish use more than 31 wait states then you will have
to edit the appropriate files manually.

How many wait states? (31) :
```

- Choose from 0 – 31 RAM sequencer wait states. The default and the value we are using is **31**. This is a good default, as it will produce an ARC that will run with almost any speed memory. For simulation purposes, or if you know that the ARC you are building will **always** use a certain speed RAM, it may be useful to set the default value for the wait states to a smaller number than the maximum 31.

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= Simulation =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

 Selection of the state of the ARC on reset is important since it is preferable
to have the system build start on reset while the core build should halt on
reset since the host effectively clears the halt bit during simulation.

 Do you wish to halt the ARC at address location 0 on reset (n) :
```

- We have entered '**n**' here, that means the ARC starts running the code at address 0 on reset. Had we selected '**y**' the ARC is halted immediately on reset.
- The default for this Y on core build and N on generic and ARCAngel builds.

```
Is your VHDL simulator Model Technologies VSystem? (y) :
```

- Argonaut Technologies we enter '**y**' as we use MTI VSystem to simulate our VHDL If you use a different simulator, select no. You may set your environment up for your simulator by editing the makefile and other support files, after the install has completed.

```
Do you wish to use the R.T.L. And SeeCode Application Link (RASCAL)? (n) :
```

- RTL And See-Code Application Link, otherwise known as RASCAL, is extra utility that connects the Metaware debugger to connect to the Modeltech simulator. This allows the debugger to control the simulation of code on the ARC system model in the MTI simulator for Hardware/Software Co-Verification.
- To include RASCAL we enter **y** and press enter.

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= Install Directory =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

Full path to ARC user working directory (/home/someone/systemarc) :
```

- At this prompt you have to specify a destination directory name (with full path).
- The path must not include /tmp_mnt/ etc.
- If the directory currently exists, it will not be destroyed, you will just be prompted for another path.
- We will select the default, a directory named "`systemarc`" created where we launched the ARC system builder.

What we have built

I-fetch address bus width	22-bit	
External address bus width	24-bit	
Register file	3-port RAM	
Fast load returns	'yes'	
Extensions	Multi-cycle barrel shifter.	
	Fast Mul/MAC.	
	Scratchpad RAM with sliding pointers.	
Technology size	0.35 µm	
Clock speed	100 MHz	
Clock skew	0.4 ns	
Users library name	'sam'	
Memory Extensions	Load/Store memory controller	
	Instruction fetch memory controller	
	Host interface for communications with ARC	
	Arbitration unit for memory accesses	
	RAM Sequencer	
Ld/St memory size	8 Kbytes.	
Type of I-Cache	Direct Mapped Cache	
I-Cache mode	Virtual Cache	
I-cache size, Max / Min	8K bytes	128 bytes
Cache line length Max / Min	8 instruction words	2 instruction words
Host communication port	Standard Sun/PC	
Arbitration Channels	Load/Store channel for external memory accesses	
	Host I/f channel for debugging and data downloading	
	Instruction fetch channel for I-cache	
	Local Scratch-pad RAM channel for burst reads/writes	
	Additional Channels (5)	
Ram Sequencer Wait states	31	
Halt ARC on Reset?	'no'	
Destination directory	`/home/user/systemarc`	
Simulator = MTI?	`'yes'`	
RASCAL?	`'yes'`	

3.3 ARCAngel

The final build is the ARCAngel. A large number of the selections on the ARCAngel build are chosen automatically due to the nature of the ARCAngel development board.

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= Fast Load Returns -=-=-=-=-=-=-=-=-=-=-=-=-=-=-

 Enable Fast load returns? (n) :
```

- Fast Load returns, if the current instruction does not use register write back then the pipeline is not stalled.
- ARCAngel development allows the use of fast return loads and so will our final design, therefore we enter 'yes'.
- For more information on Fast load returns refer to the release notes.

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= Extensions Builder =-=-=-=-=-=-=-=-=-=-=-=-=-=-

Select the extensions to be added to the ARC

   1.  Continue
   2.    32x32 Barrel shifter - rotate, arithmetic and logical shifts - 1 cycle
   3.    32x32 Barrel shifter - rotate, arithmetic and logical shifts - Multi-cycle
   4.    Small multi-cycle 32x32 Multiply
 N/A     Faster (& Larger) 32x32 Multiply
 N/A     16x16 Multiply/Multiply Accumulate function with 36 bit accumulator - Fast
   7.    16x16 Multiply/Multiply Accumulate function with 36 bit accumulator - Slow
   8.    Swap function - swaps upper and lower 16 bits of 32 bit word
   9.    MIN/MAX function - 32 bit precision
  10.    Normalise function - finds first bit set in 32 bit word
  11.  * 24-bit Timer which counts to limit and then generates an interrupt
 N/A     A Scratchpad RAM
 N/A     A Scratchpad RAM with Sliding Pointers

Note the multi-cycle barrel shifter has been selected along with the 24-bit timer.
These have been selected as part of the standard Altera configuration, however,
the system will still function correctly without them.

Please select :
```

- The above menu shows all the standard extensions, you can select lots of different combinations depending on your needs.
- Note: The scratchpad RAM and the single cycle barrel shifter are too large to fit on the current ARCAngel FPGA, as are the fast 32x32 multiplier and the fast MUL/MAC. Also a build with all extensions selected will not fit on the current FPGA.
- The 24-bit Timer is automatically selected, as it is a common selection on an ARCAngel build. We are using this extension however you can deselect it by entering the corresponding number. We will also select the Multi-Cycle Barrel shifter and the Small 32x32 multiply. Entering 3 then 4 does this.
- Note you cannot select the fast type of Mul/MAC, in an install, if you attempt to select it the ARC system builder prompts you for another response.
- We will now enter 1 to continue.

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= Ifetch Modes =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

 Select the instruction Cache Modes.

   1. Continue
   2. *Standard Cache with debug and cache bypass capability.
   3.  Virtual Cache
   4.  Mixed Code RAM / Icache space & Cache line lock

Please select :
```

- The first thing the script needs to know is what kind of cache do you want. It's possible to have a :
    1. Standard Cache
    2. Virtual cache allows a user to dynamically resize the cache in software.
    3. Mixed Code RAM / Icache space with Line locking

Selecting 3 allows standard Cache with the extra features of Code RAM and Line locking.

For more information on Instruction cache modes refer to the ARC interface manual.
NOTE: Standard Cache and Virtual Cache are mutually exclusive, and Mixed Code RAM is a superset of Standard Cache.

```
Please select the norm size of the Direct Mapped Instruction Cache.

   1.  0.5k bytes, 128 words
   2.  1k bytes, 256 words
   3.  2k bytes, 512 words
   4.  4k bytes, 1024 words
   5.  8k bytes, 2048 words
   6.  16k bytes, 4096 words

Please select (3) :
```

- The Direct Mapped Instruction cache is user definable, using the script you can input select sized from 512 bytes (128 words), up to 16k bytes (4k words). Other sizes (larger or smaller) can be manually instantiated but for our test 4K bytes (selection **4**) is sufficient.

The ARCAngel has onboard tag-RAM but uses external memory for the cache data-RAMS, so selecting larger cache sizes does not drastically affect the amount of RAM resources available on the Altera device.

```
Cache line length

  1.  2 instruction words
  2.  4 instruction words
  3.  8 instruction words
  4.  16 instruction words
  5.  32 instruction words
Please select (3) :
```

- The cache line length is the number of words fetched when a cache miss occurs. The script has options for 2, 4, 8, 16, or 32 word lines, the default being **8 words.** Entering **2** will give a cache line length of 4 words, suitable for our test.

Cache size, line length and external address all determine the size of the tag RAM. Larger cache → more tag words fewer bits. Longer line length → less tag words more tag bits.

```
-============== Host Interface Communications ================-

Select from available host communications ports.

   1.  Standard PC/SUN Parallel Communications Port
   2.  JTAG Communications Port

Please select (1) :
```

- Host port for I/O communications can be either a standard PC/SUN or a JTAG communications port. We are selecting **2**, JTAG, as we wish to have an ARCAngel that can communicate to a range of host machines via the industry standard JTAG.

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= Simulation =-=-=-=-=-=-=-=-=-=-=-=-=-=

 Selection of the state of the ARC on reset is important since it is preferable
to have the system build start on reset while the core build should halt on
reset since the host effectively clears the halt bit during simulation.

 Do you wish to halt the ARC at address location 0 on reset (n) :
```

- We have entered '**n**' here, that means the ARC starts running the code at address 0 on reset. Had we selected 'y' the ARC is halted immediately on reset.
- The default for this Y on core build and N on generic and ARCAngel builds.

```
Is your VHDL simulator Model Technologies VSystem? (y) :
```

- At Argonaut Technologies we enter 'y' as we use MTI VSystem to simulate our VHDL If you use a different simulator, select no. You may set your environment up for your simulator by editing the makefile and other support files, after the install has completed.

```
Do you wish to use the R.T.L. And SeeCode Application Link (RASCAL)? (n) :
```

- RTL And See-Code Application Link, otherwise known as RASCAL, is extra utility that connects the Metaware debugger to connect to the Modeltech simulator. This allows the debugger to control the simulation of code on the ARC system model in the MTI simulator for Hardware/Software Co-Verification.
- The default of **n** is selected by simply pressing enter.

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-= Install Directory =-=-=-=-=-=-=-=-=-=-=-=-=-=-=

Full path to ARC user working directory (/home/someone/arcangel) :
```

- At this prompt you have to specify a destination directory name (with full path).
- The path must not include /tmp_mnt/ etc.
- If the directory currently exists, it will not be destroyed; you will just be prompted for another path.
- We will select the default; a directory named "`arcangel`" created where we launched the ARC build script.

NOTE: Even though the bus width is set to 26-bits, the memory model for a ARCAngel build simulation is 512Kbytes.

What we have built

Fast Load returns?	'yes'
Extensions	Multi-cycle barrel shifter.
	24-bit timer.
	Small 32x32 multiplier
Direct mapped instruction cache size	4K bytes
Cache line length	4 instruction words
Host interface controller	JTAG
Ram Sequencer	SRAM 0 Wait states
Halt ARC on Reset?	'no'
Destination directory	`/home/user/systemarc`
Simulator = MTI?	`'yes'`
RASCAL?	`'no'`

```
Is your VHDL simulator Model Technologies VSystem? (y) :
```

- At Argonaut Technologies we enter 'y' as we use MTI VSystem to simulate our VHDL If you use a different simulator, select no. You may set your environment up for your simulator by editing the makefile and other support files, after the install has completed.

```
Do you wish to use the R.T.L. And SeeCode Application Link (RASCAL)? (n) :
```

- RTL And See-Code Application Link, otherwise known as RASCAL, is extra utility that connects the Metaware debugger to connect to the Modeltech simulator. This allows the debugger to control the simulation of code on the ARC system model in the MTI simulator for Hardware/Software Co-Verification.
- The default of **n** is selected by simply pressing enter.

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-= Install Directory =-=-=-=-=-=-=-=-=-=-=-=-=-=

Full path to ARC user working directory (/home/someone/arcangel) :
```

- At this prompt you have to specify a destination directory name (with full path).
- The path must not include /tmp_mnt/ etc.
- If the directory currently exists, it will not be destroyed; you will just be prompted for another path.
- We will select the default; a directory named "`arcangel`" created where we launched the ARC build script.

NOTE: Even though the bus width is set to 26-bits, the memory model for a ARCAngel build simulation is 512Kbytes.

What we have built

Fast Load returns?	'yes'
Extensions	Multi-cycle barrel shifter.
	24-bit timer.
	Small 32x32 multiplier
Direct mapped instruction cache size	4K bytes
Cache line length	4 instruction words
Host interface controller	JTAG
Ram Sequencer	SRAM 0 Wait states
Halt ARC on Reset?	'no'
Destination directory	`/home/user/systemarc`
Simulator = MTI?	`'yes'`
RASCAL?	`'no'`

3.4 Core Verification system

The Core verification is a build which creates an ARC system and appropriate test files to verify that the ARC VHDL behaves as described in the ARC Programmers Reference manual.

NOTE: Even though the bus width is set to 26-bits, the memory model for a core build simulation is 128Kbytes.

```
=-=-=-=-=-=-=-=-=-=-=-= Core Register File Selection =-=-=-=-=-=-=-=-=-=-=-=-

 The 32x32 register file can be implemented using a synthesised array
of d-latches, or by using a three-port (2 read, 1 write) synchronous
RAM cell.

Use a synchronous 2r1w RAM for the register file? (y) :
```

- For the 32 general purpose registers r0-r31 you can either use :-
  (y) Synchronous 3-port (2r1w) RAM, where a fast write-through not required, or
  6. (n) Synthesised arrays of d-latches that require a write pulse to be generated.
- Standard synthesis script assumes a 90°-delayed clock signal.

We've got a synchronous 3-port RAM and so we select **y**.

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= Technology Selection =-=-=-=-=-=-=-=-=-=-=-=-=

Feature size (drawn) of technology in use

   1.  0.35um
   2.  0.5um
   3.  0.65um
   4.  0.8um
   5.  1.0um

Please select (5) :
```

The feature size provides timings for DATA / TAG ram simulation & synthesis models and the core registers. Our chosen technology is 0.65µm so press **3** to continue.

```
Target clock speed in MHz (50)
```

- We will use the default of **50Mhz** as we have specified 0.65 µm technology.

```
Clock skew for synthesis (+/-0.4ns) :
```

- The default of **+/- 0.7ns** will be used for this walkthrough.

```
Library for extensions logic? (user) :
```

- Here you can specify a name for your user's library. This name will be used during the install when creating two directories and copying subsequent files to these directories. The string entered is arbitrary so we will enter 'tom'.

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= Simulation =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

 Selection of the state of the ARC on reset is important since it is preferable
to have the system build start on reset while the core build should halt on
reset since the host effectively clears the halt bit during simulation.

 Do you wish to halt the ARC at address location 0 on reset (y) :
```

- We have entered 'y' here, that means the ARC is halted immediately on reset. Had we selected 'n' the ARC starts running the code at address 0 on reset.
- The defaults are 'Y' for a core build and 'N' for generic and ARCAngel builds.

```
Note you will have to set the status register to the appropriate start address
and clear the halt bit through the host interface before running the simulation.
You can manually edit "arc_start" in  xaux_regs.vhdl so that the ARC is running
following a reset.
```

- Having selected 'y' the above information message is displayed.

```
Is your VHDL simulator Model Technologies VSystem? (y) :
```

At Argonaut Technologies we enter 'y' as we use MTI VSystem to simulate our VHDL If you use a different simulator, select no. You may set your environment up for your simulator by editing the makefile and other support files, after the install has completed.

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= Install Directory =-=-=-=-=-=-=-=-=-=-=-=-=-=-=

Full path to ARC user working directory (/home/someone/verifyarc) :
```

- At this prompt you have to specify a destination directory name (with full path).
- The path must not include /tmp_mnt/ etc.
- If the directory currently exists, it will not be destroyed, you will just be prompted for another path.
- We will select the default, a directory named "`verifyarc`" created where we launched the ARC system builder.

What we have built

Technology	
Feature Size	0.65um
Clock Speed in MHz	50 MHz
Clock Skew in ns	+/- 0.7 ns
Halt ARC on Reset?	'yes'
Destination directory	`/home/user/verifyarc`
Simulator = MTI?	`'yes'`

4 - Simulation and Synthesis

4.1 What is generated

Once the script has completed the install, the base case / ARCAngel build generates files for simulation and synthesis. The Generic builder can generate files for simulation provided all the required memory extensions are selected.

For all types of build the VHDL that the user can modify has been included in the directory `'./vhdl'`

The directories for Model Tech VSystem simulator are: -

mti_user/	MTI Libraries
mti_ARC/	"
mti_DW01/	"
mti_DW02/	"
anz/	Target Directory for all analysed files.

As default the test code is linked to the tests: -

Core/Generic & ARCAngel builds : init_mem.hex

Core Verification build : testscript.hcs*

Type of Tests	Hex Directories
ALU and interrupts	Core_b
Extension Core Interfaces	Core_x
Extension interfaces	Ext_x
Host interface	hm
Auxiliary logic	Aux_x

*Refer to Core Verification Overview document for further information.

Base Case

For simulation a makefile is auto-generated to compile all the selected VHDL for the MTI VSystem simulator. Type 'make' to build database, see makefile header for information on using the makefile with other simulators.

To simulate, simply type 'coretest'.

This runs the Model Technologies simulator from the command line. The behaviour of the ARC core is displayed in the form of a pipeline diagram (SEE FIGURE 3) that shows the state of the processor as it processes each instruction. The user is made aware when host interface accesses are taking place, in addition to Load/Store from external memory.

NOTE: The code will be executed twice for the default set-up since it is run in normal mode and also in single step mode where each instruction is single stepped through the pipeline. The single step feature is only executed upon successful conclusion of the test code in normal operational mode.

Upon conclusion the user is made aware whether the test being simulated passed or failed.

The synthesis scripts and support files are kept in the following directories: -

scripts/	synthesis scripts
USER/	Synopsys VHDL libraries
ARC/	Synopsys VHDL libraries

All the files that are modifiable are compiled to the user library, except for the file `extutil.vhdl` which is compiled to the ARC library.

To synthesise type 'syn_arc', which runs the synthesis scripts for Synopsys Design Compiler.

A README.txt file is also included which outlines how the user can simulate and synthesise the current base case installation.

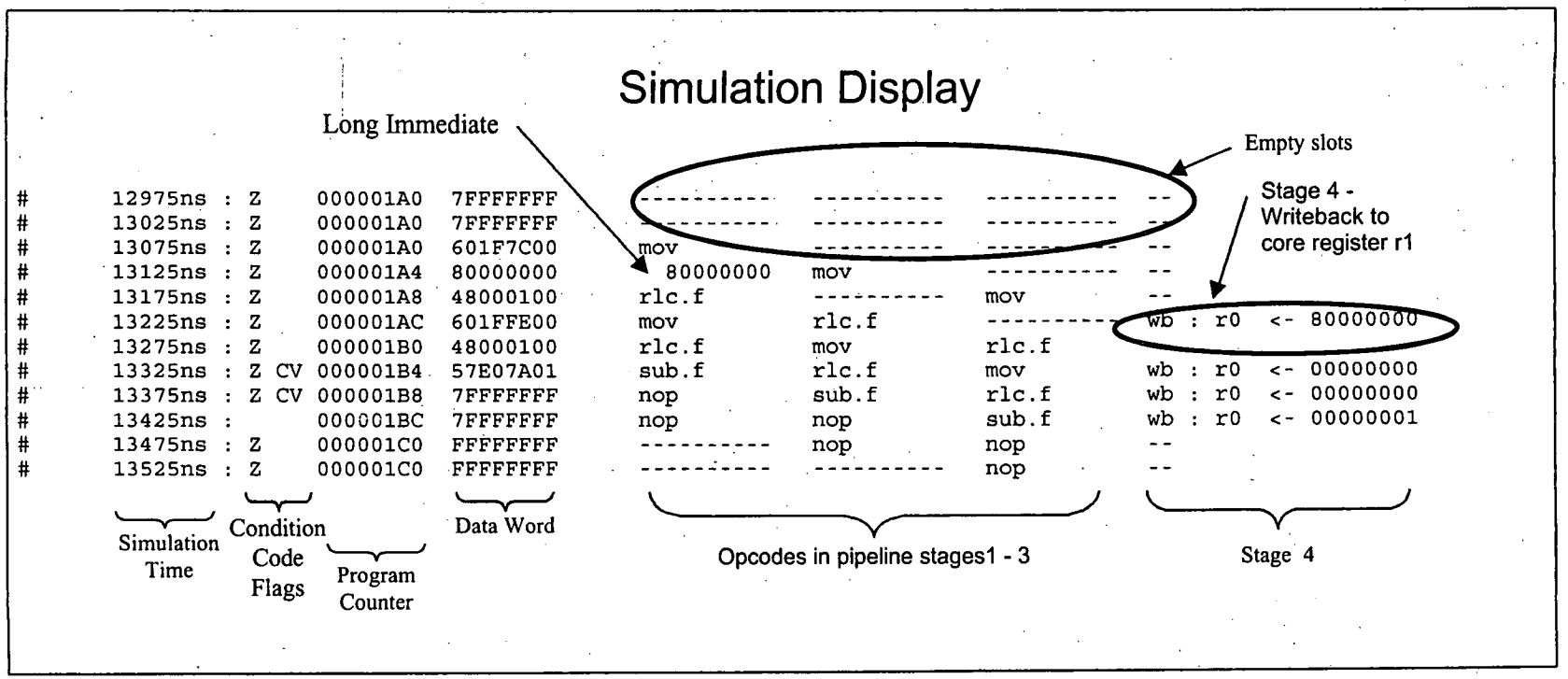


Figure 3 Example of Pipeline diagram.

Generic build & ARCAngel builds

The ARCAngel build allows the designer to simulate and synthesise an ARC to the FPGA on the ARCAngel development board so that extensions and other features that have been added can be debugged.
The generic build is similar however memory extensions can be added in addition to the core extensions. This build allows the designer to simulate most variations although certain combinations provide 'Dummy' blocks for the designer to add their own functionality. The generic system can be synthesised from auto-generated scripts for the Synopsys Design Compiler.

To simulate, simply type 'asictest'.

Note that in an ARCAngel build an additional directory is also inserted as part of the build, i.e. "aa_chip". Once synthesis has been completed this will contain all the information with regards to generating data for the Altera flex10K100 FPGA on the ARCAngel development board. The arcangel_chip.sof configuration file is created by the command line parameters for maxplus2, set in the syn_arc executable.

The raw binary version of the ARCAngel system requires the maxplus2 GUI, before blasting the information onto the development board. For more information on how to do this please refer to the Development Board (ARCAngel) documentation.

To run your own code in the Generic or ARCAngel builds
First - we need to have produced a Mentor-quicksim format HEX file from the Metaware tools - This has already been done for basecase and extensions test code, since the basecase memory models read these files. The system builds are also capable of reading hex files, however, the user must be aware that test code generated must

Upon conclusion (when the ARC has been halted) the user is made aware whether the test being simulated passed or failed. This is accomplished by reading the contents of the Status register via the host port to check whether the Zero flag has been set.

Core Verification system build

The Core verification system defaults to settings that allow the user to simulate, synthesise and verify the ARC core.

For more information see the ARC core verification overview document.

4.2 Running Extensions test code

When an installation has been completed, the simulation is set up to run the basecase test code. The VHDL reads a file '`init_mem.hex`' from the working directory that contains a memory image of the test program.

This file is actually a symbolic link to the file
`$ARCHOME/arc/src/metaware/coretest.hex`

To run test code for extensions logic, it is necessary to alter the link in the working directory to point to an alternative memory image file. Pre-compiled memory image files (`.hex`) for testing the extensions are located in the directory `$ARCHOME/exts/src/metaware`. Listing files (`.lis`) are stored along with the source code and memory image files.

Note: The addresses given in the .lis files are relative to the start of the section. Hence when attempting to match an address from a simulation to the source line which created it subtract 0x100 from the real address to find the instruction in the .lis file. The .dump files show real addresses and disassembly, but none of the original source lines.

Compiling the extensions test source code (testxalu.s) with the script mw.bat created these files.

Memory image	Tests
x_bshift.hex	32 bit barrel shift/rotate block
x_minmax.hex	MIN and MAX instructions
x_mul64.hex	32 x 32 scoreboard multiplier with 64-bit result
x_mulmac.hex	16-bit Multiply/Multiply-accumulate function with 36-bit accumulator
x_norm.hex	Normalise (find-first-bit) instruction
x_swap.hex	SWAP instruction
x_ldstram.hex	Local LD/ST RAM
x_timer.hex	24-bit timer counter
x_scratch.hex	Scratch Pad RAM.
x_slide.hex	Scratch Pad RAM with sliding pointers.
x_all.hex	All extensions together.

For example, to test an ARC configured with the barrel shifter block, do the following from the working directory:

```
rm init_mem.hex
ln -s $ARCHOME/exts/src/metaware/x_bshift.hex init_mem.hex
```

Now run the VHDL simulation.

To replace the link to the basecase test code:

```
rm init_mem.hex
ln -s $ARCHOME/arc/src/coretest.hex init_mem.hex
```

Single step tests (For Core build)

The standard testbench runs the code provided in `init_mem.hex`, and then runs it again, single stepping each of the instructions. Editing the file vhdl/memcon2.vhdl can disable this. Altering the signal assignment of no_step_test to supply the value '1' does this.

Tests (For Generic/ARCAngel build)

The standard testbench runs code provided in `init_mem.hex`. For the builds which inlcude the PC port modifying the variable do_pc_test to '1' in glue.vhdl can also set the PC test, provided that the hex file `$ARCHOME/exts/src/metaware/aa.hex` is used. Note that single step instruction mode for test code can be enabled by modifying the variable no_step_test to '0' in glue.vhdl. The single stepping feature is executed only when the test code has been successfully run in normal operation mode.

The JTAG communications port is tested in a similar manner to the PC port apart from the fact that the testbench determines that if a JTAG port is present (via jtag_model), then tests for the communication port should be executed. Note that single step instruction mode for test code can be enabled by modifying the variable no_step_test to '0' in glue.vhdl. The single stepping feature is executed only when the test code has been successfully run in normal operation mode.

Both of the host models for the PC and JTAG port implement HMSL (Host Model Script Language) to program the behaviour of the communication ports. The system builder links the working directory to the script at `$ARCHOME/system/comms/hmsl/host_program.hmsl`.

4.3 What is customer modifiable

To create further additions to the test the user can use the METAWARE HighC compiler. The core build can read the output from the METAWARE tool set directly, i.e. HEX format files. The ARCAngel/generic builds also read the HEX format files output by HighC.

Cache test code

Test code has been provided to allow the cache logic to be tested. It must be edited by the user to set the cache size which is in use.

Make a subdirectory under your working area, and copy the appropriate files into it:

```
llama> mkdir src
llama> cd src
llama> cp $ARCHOME/exts/src/metaware/cache.* .
llama> cp $ARCHOME/exts/src/metaware/macros.s .
llama> chmod +w cache.s
```

Now edit the test code file cache.s to set the CACHESIZE variable to the appropriate value, and recompile the code. This will require a certain amount of copying files between PC and Sun systems if the Metaware tools are not available on your Sun system.

```
llama> cache.bat
```

You may now either link cache.hex to init_mem.hex for your specific build, i.e. core or system.

The designer can edit the files in the VHDL directory to modify or add features to the extension core logic:-

xalu.vhdl	Extension alu functions.
xdefs.vhdl	Extension instruction opcodes, auxiliary registers and other constants.
xrctl.vhdl	Extension control logic for pipeline flow.
xaugs_regs.vhdl	Auxiliary registers.
xcore_regs.vhdl	Core registers.

There are memory extension files that can be edited. Depending on the user's selection, files that can be edited are as follow: -

sram_seq.vhdl	RAM sequencer.
mc_sys.vhdl	Memory arbitration unit.
mc_arc.vhdl	Ld/st memory controller.
dmcc.vhdl	Direct mapped cache controller.
i_fetch.vhdl	Dummy instruction fetch.
miu.vhdl	External memory i/f unit.

The synthesis scripts and support files can also be edited: -

.synopsys-dc.setup	Technology-specific setup
user_synopsys_dc.setup	ARC-specific setup. Included by above
scripts/elaborate.dc	Elaboration of VHDL.
scripts/analyse.dc	Analysis of VHDL.
scripts/do_all.dc	Complete synthesis of design.

Hierarchy Generation

If additional files are added to the hierarchy then the makefile must be modified to reflect this. However to update the structure simply edit the files in the 'dat' directory. The files to edit are board.hier (or apex.hier for core build) and the library.list file.

See the "Automatic Hierarchy generator' document for more details.

Appendix

Menu structure : -

See pages diagram

Command line parameters : -

NAME

system_builder - builds ARC VHDL files for simulation & synthesis.

SYNOPSYS

system_builder ARCgui [-parameters]

DESCRIPTION

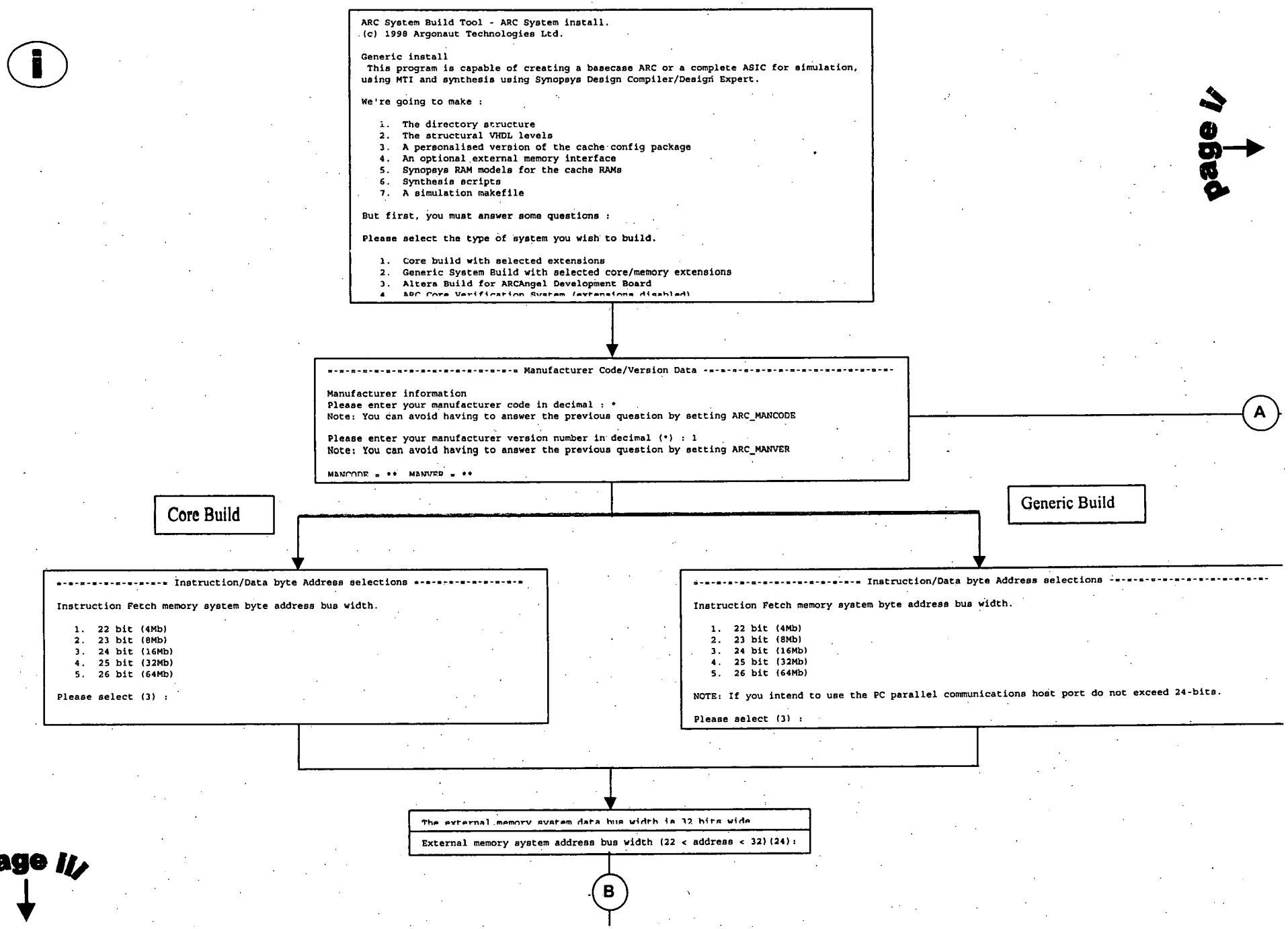
The system builder

REQUIRED PARAMETERS

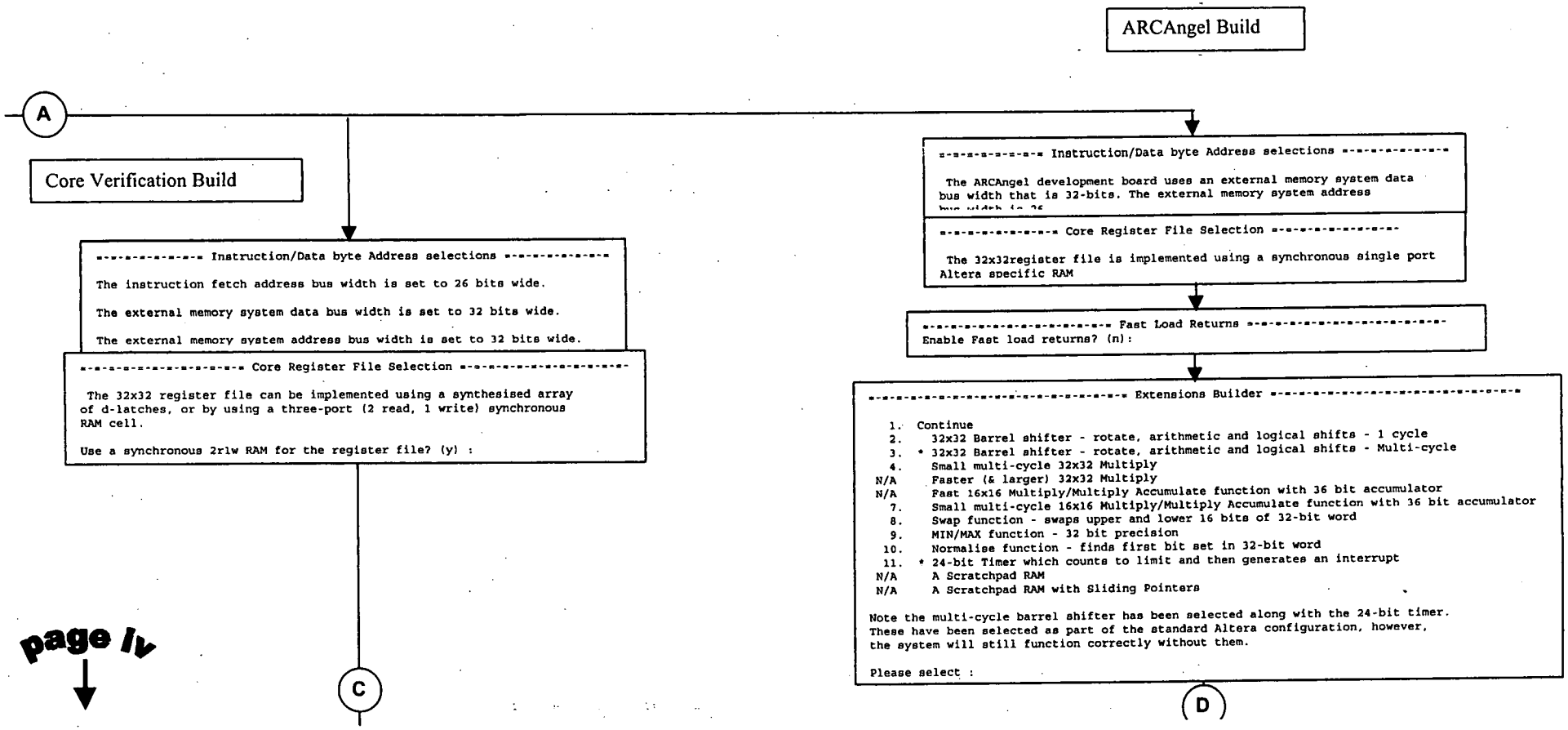
-build_type	[value]	core, generic, altera, core_verification
-ifetch_addr	[value]	19 - 26
-ext_addr	[value]	19 - 32
-tech	[value]	Feature size. e.g. 0.35um
-ck_speed	[value]	Clock speed in MHz e.g. 100
-ck_skew	[value]	Clock Skew +/- value in ns.
-ext_lib	[value]	Name to be used for ext. library.
-local_ram	[value]	Amount of Local load/st RAM
-ifetch_system	[value]	dummy, Direct Mapped Cache
-cache_size	[value]	Cache size
-cache_line_length	[value]	Cache Line lenght
-add_chan	[value]	Additional channels
-wait_states	[value]	Ram sequencer wait states
-install_dir	[path]	Where destination dir is to be created
-install_name	[dir name]	Name of the destination directory

OPTIONAL PARAMETERS

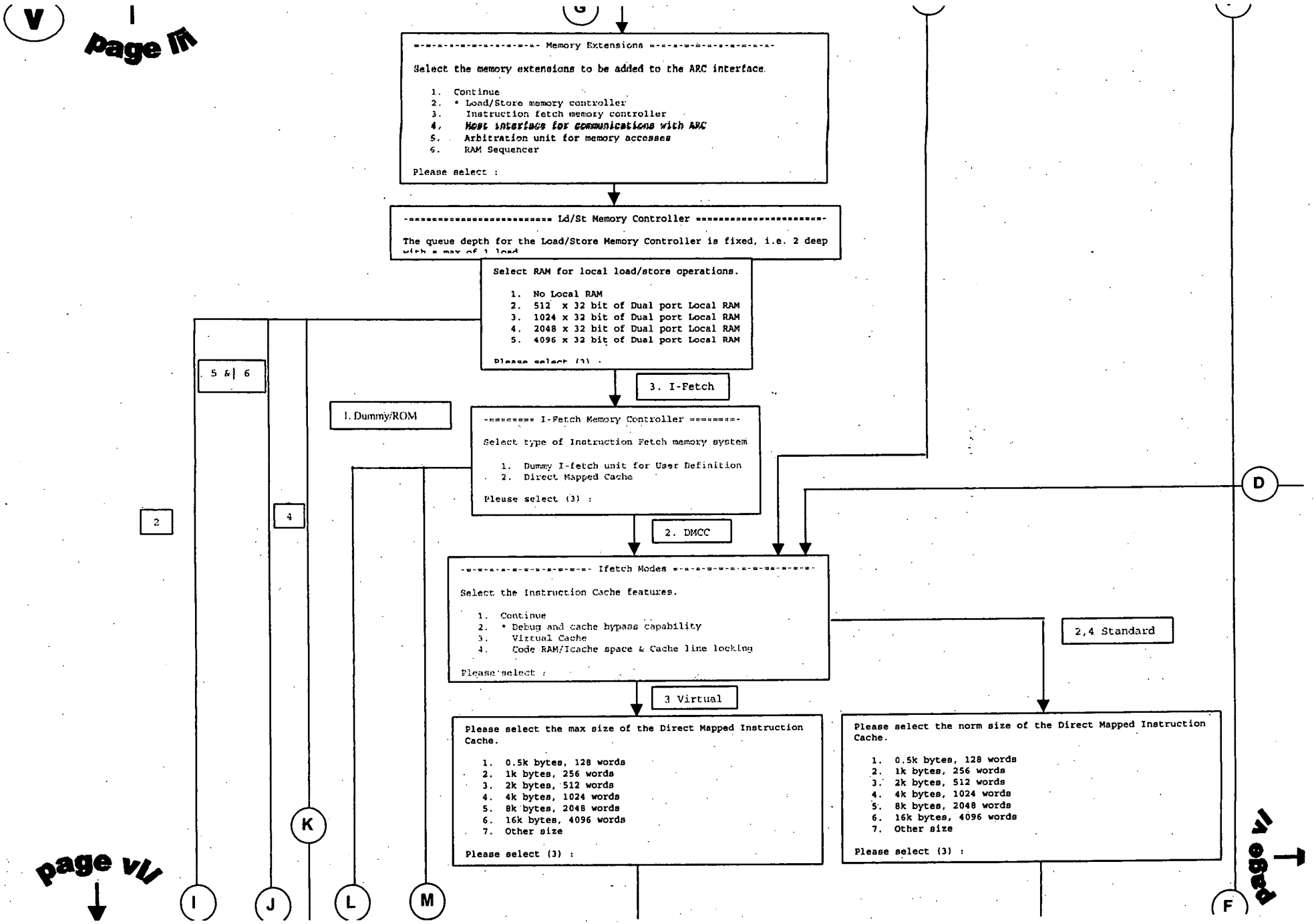
-register_file	Use synchronous RAM cell register file.	
-cache	Cache enabled	
-ldst_cont	load/store controller	
-hostif_cont	host interface controller	
-host_port	[value]	PC,dummy
-ifetch_cont	I-fetch memory controller	
-arbitrator	Memory Arbitrator	
-ram_seq	Ram sequencer	
-sram_chan	Ram Sequencer Memory Arbitrator channel	
-halt	Halt the ARC on reset	
-mti	Compile for use with MTI Vsystem.	


ARC System Build Tool - ARC System install.
(c) 1998 Argonaut Technologies Ltd.
Generic install
This program is capable of creating a basecase ARC or a complete ASIC for simulation, using MTI and synthesis using Synopsys Design Compiler/Design Expert.
We're going to make :
1. The directory structure
2. The structural VHDL levels
3. A personalised version of the cache config package
4. An optional external memory interface
5. Synopsys RAM models for the cache RAMs
6. Synthesis scripts
7. A simulation makefile
But first, you must answer some questions :
Please select the type of system you wish to build.
1. Core build with selected extensions
2. Generic System Build with selected core/memory extensions
3. Altera Build for ARCAngel Development Board
4. ARC Core Verification System (extensions disabled)
page i
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= Manufacturer Code/Version Data -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
Manufacturer information
Please enter your manufacturer code in decimal : *
Note: You can avoid having to answer the previous question by setting ARC_MANCODE
Please enter your manufacturer version number in decimal (*) : 1
Note: You can avoid having to answer the previous question by setting ARC_MANVER
MANCODE = ** MANVER = **
A
Core Build
Generic Build
-=-=-=-=-=-=-=-=-= Instruction/Data byte Address selections -=-=-=-=-=-=-=-=-=
Instruction Fetch memory system byte address bus width.
1. 22 bit (4Mb)
2. 23 bit (8Mb)
3. 24 bit (16Mb)
4. 25 bit (32Mb)
5. 26 bit (64Mb)
Please select (3) :
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= Instruction/Data byte Address selections -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
Instruction Fetch memory system byte address bus width.
1. 22 bit (4Mb)
2. 23 bit (8Mb)
3. 24 bit (16Mb)
4. 25 bit (32Mb)
5. 26 bit (64Mb)
NOTE: If you intend to use the PC parallel communications host port do not exceed 24-bits.
Please select (3) :
The external memory system data bus width is 32 bits wide
External memory system address bus width (22 < address < 32)(24):
B
page iii

page i



page iv


V
page IV
G
Memory Extensions
Select the memory extensions to be added to the ARC interface.
1. Continue
2. * Load/Store memory controller
3. Instruction fetch memory controller
4. Host interface for communications with ARC
5. Arbitration unit for memory accesses
6. RAM Sequencer
Please select :
Ld/St Memory Controller
The queue depth for the Load/Store Memory Controller is fixed, i.e. 2 deep
Select RAM for local load/store operations.
1. No Local RAM
2. 512 x 32 bit of Dual port Local RAM
3. 1024 x 32 bit of Dual port Local RAM
4. 2048 x 32 bit of Dual port Local RAM
5. 4096 x 32 bit of Dual port Local RAM
3. I-Fetch
1. Dummy/ROM
I-Fetch Memory Controller
Select type of Instruction Fetch memory system
1. Dummy I-fetch unit for User Definition
2. Direct Mapped Cache
Please select (3) :
2. DMCC
Ifetch Modes
Select the Instruction Cache features.
1. Continue
2. * Debug and cache bypass capability
3. Virtual Cache
4. Code RAM/Icache space & Cache line locking
Please select :
2,4 Standard
3 Virtual
Please select the max size of the Direct Mapped Instruction Cache.
1. 0.5k bytes, 128 words
2. 1k bytes, 256 words
3. 2k bytes, 512 words
4. 4k bytes, 1024 words
5. 8k bytes, 2048 words
6. 16k bytes, 4096 words
7. Other size
Please select (3) :
Please select the norm size of the Direct Mapped Instruction Cache.
1. 0.5k bytes, 128 words
2. 1k bytes, 256 words
3. 2k bytes, 512 words
4. 4k bytes, 1024 words
5. 8k bytes, 2048 words
6. 16k bytes, 4096 words
7. Other size
Please select (3) :
5 & 6
2
4
D
K
I
J
L
M
F
page VII
page VI

page vii

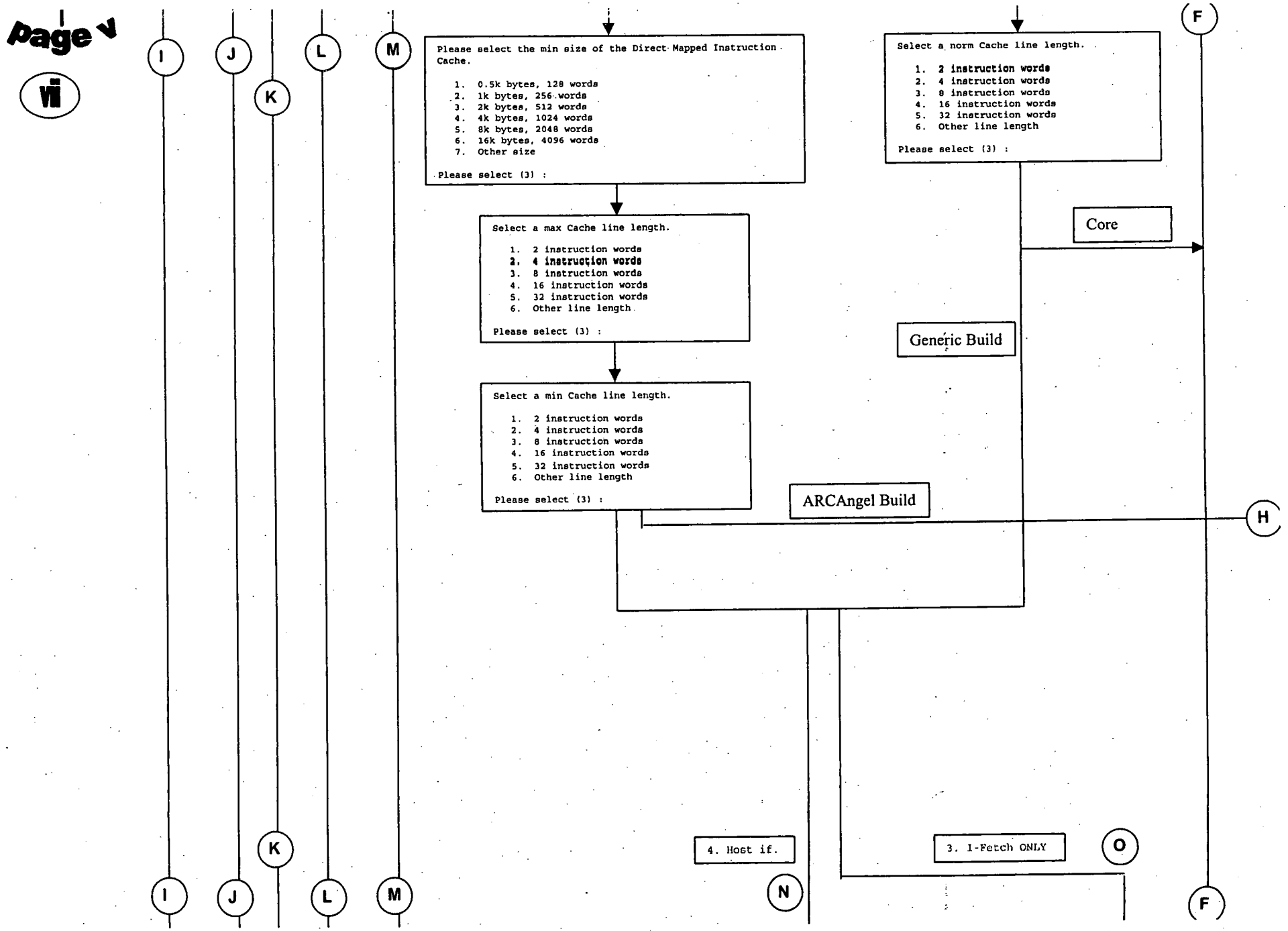
page v

page iv

vi

D

D


Page
vi
I
J
K
L
M
Please select the min size of the Direct Mapped Instruction Cache.
1. 0.5k bytes, 128 words
2. 1k bytes, 256 words
3. 2k bytes, 512 words
4. 4k bytes, 1024 words
5. 8k bytes, 2048 words
6. 16k bytes, 4096 words
7. Other size
Please select (3) :
Select a max Cache line length.
1. 2 instruction words
2. 4 instruction words
3. 8 instruction words
4. 16 instruction words
5. 32 instruction words
6. Other line length
Please select (3) :
Select a min Cache line length.
1. 2 instruction words
2. 4 instruction words
3. 8 instruction words
4. 16 instruction words
5. 32 instruction words
6. Other line length
Please select (3) :
Select a norm Cache line length.
1. 2 instruction words
2. 4 instruction words
3. 8 instruction words
4. 16 instruction words
5. 32 instruction words
6. Other line length
Please select (3) :
F
Core
Generic Build
ARCAngel Build
H
4. Host if.
N
3. I-Fetch ONLY
O
F

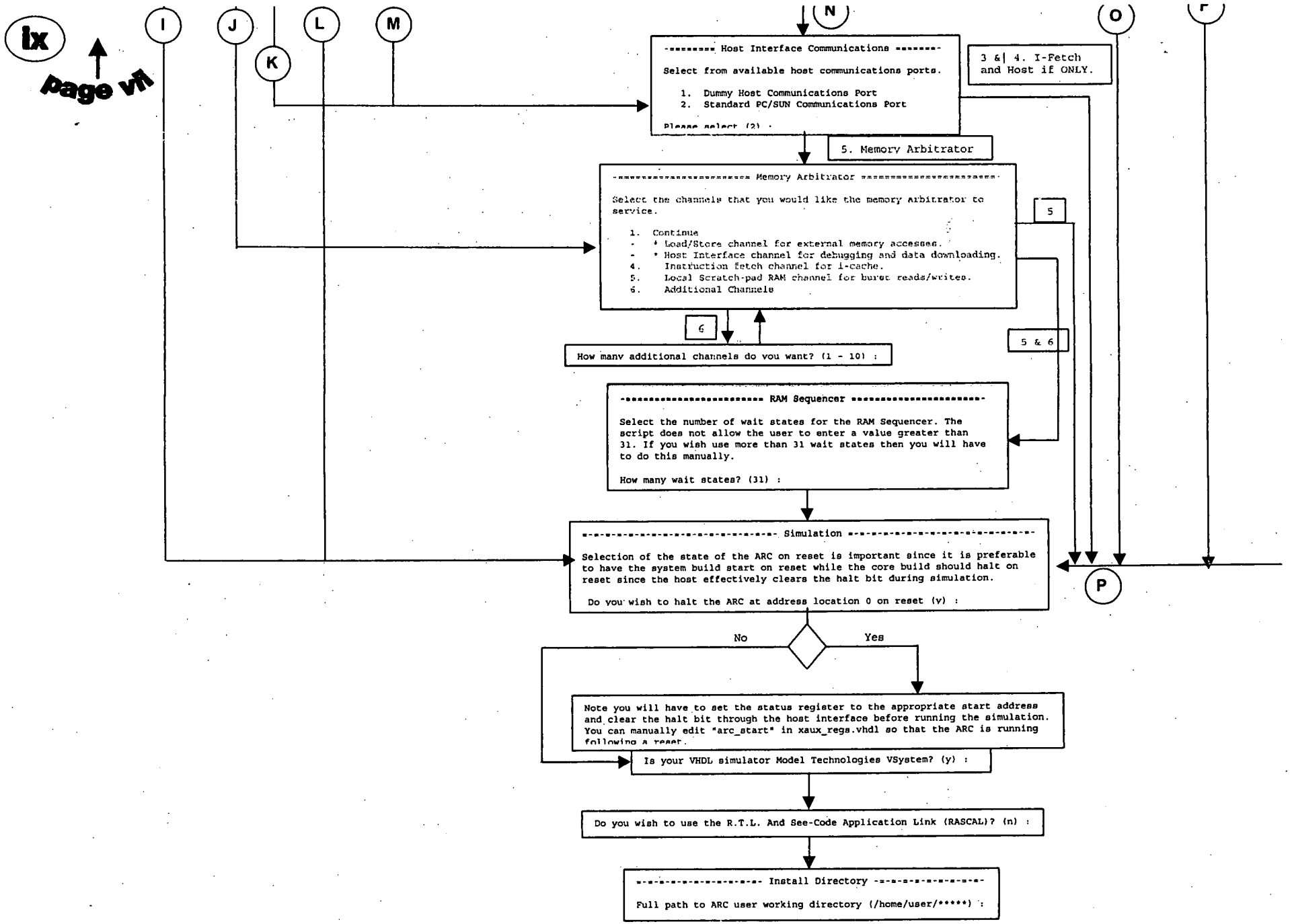
page vi

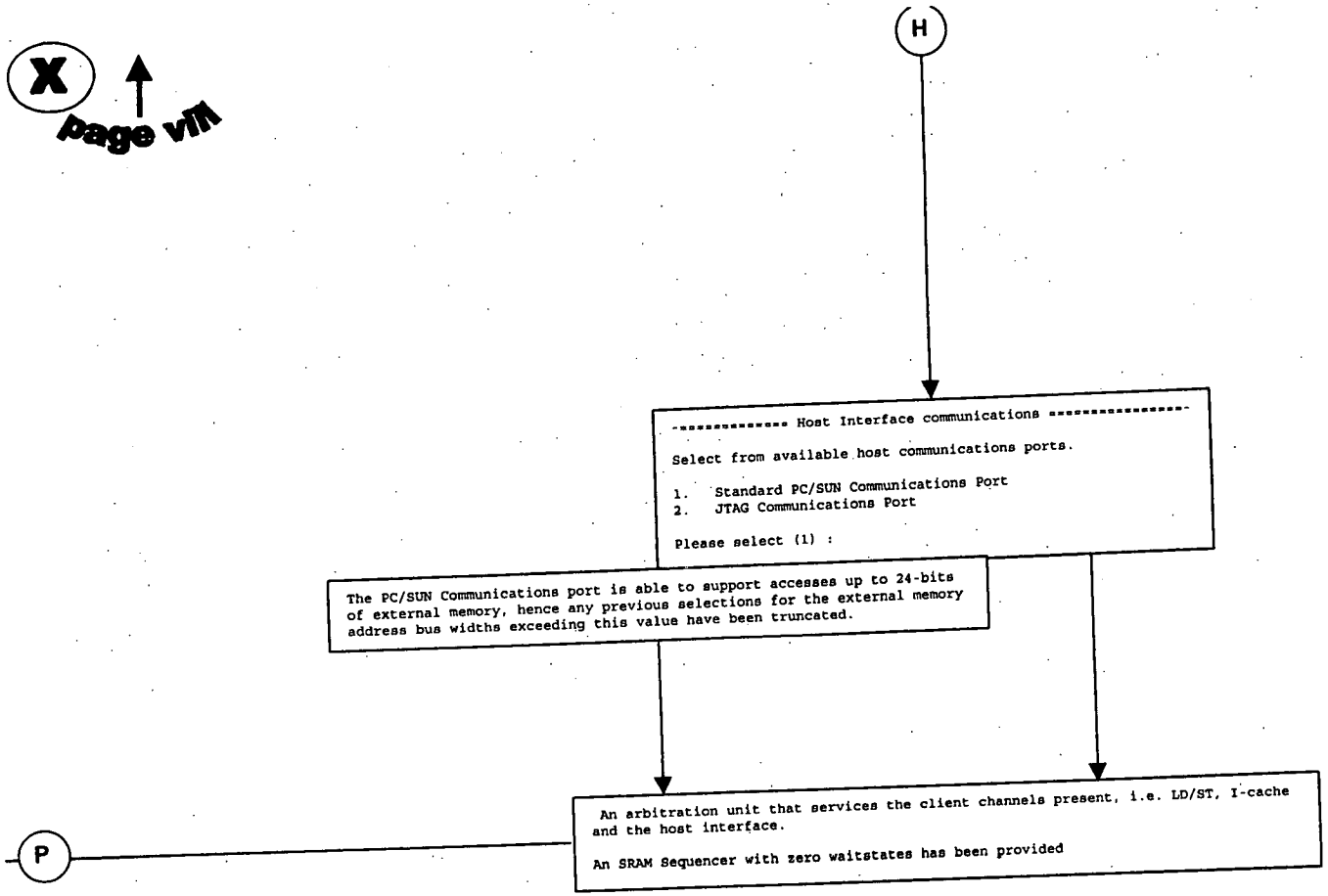
H

H

page vii

VIII


ix
Page vii
I
J
K
L
M
N
O
P
-------- Host Interface Communications --------
Select from available host communications ports.
1. Dummy Host Communications Port
2. Standard PC/SUN Communications Port
Please select (2) :
3 & 4. I-Fetch and Host if ONLY.
5. Memory Arbitrator
---------- Memory Arbitrator ----------
Select the channels that you would like the memory arbitrator to service.
1. Continue
* Load/Store channel for external memory accesses.
* Host Interface channel for debugging and data downloading.
4. Instruction fetch channel for i-cache.
5. Local Scratch-pad RAM channel for burst reads/writes.
6. Additional Channels
5
6
5 & 6
How many additional channels do you want? (1 - 10) :
---------- RAM Sequencer ----------
Select the number of wait states for the RAM Sequencer. The script does not allow the user to enter a value greater than 31. If you wish use more than 31 wait states then you will have to do this manually.
How many wait states? (31) :
---------- Simulation ----------
Selection of the state of the ARC on reset is important since it is preferable to have the system build start on reset while the core build should halt on reset since the host effectively clears the halt bit during simulation.
Do you wish to halt the ARC at address location 0 on reset (y) :
No
Yes
Note you will have to set the status register to the appropriate start address and clear the halt bit through the host interface before running the simulation. You can manually edit "arc_start" in xaux_regs.vhdl so that the ARC is running following a reset.
Is your VHDL simulator Model Technologies VSystem? (y) :
Do you wish to use the R.T.L. And See-Code Application Link (RASCAL)? (n) :
---------- Install Directory ----------
Full path to ARC user working directory (/home/user/*****) :


X
page vii
H
---------- Host Interface communications ----------
Select from available host communications ports.
1. Standard PC/SUN Communications Port
2. JTAG Communications Port
Please select (1) :
The PC/SUN Communications port is able to support accesses up to 24-bits of external memory, hence any previous selections for the external memory address bus widths exceeding this value have been truncated.
An arbitration unit that services the client channels present, i.e. LD/ST, I-cache and the host interface.
An SRAM Sequencer with zero waitstates has been provided
P
page 1x

page i
iii
page iv
page v